Joonas Laitio

Semantic Web Data Quality Control

Department of Media Technology

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo, 26.09.2011

Thesis supervisor and instructor:

Prof. Eero Hyvönen

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Joonas Laitio

Title: Semantic Web Data Quality Control

Date: 26.09.2011 Language: English Number of pages:6+63

Degree Programme of Automation and Systems Technology
Department of Media Technology

Professorship: T-75

Supervisor and instructor: Prof. Eero Hyvönen

Data quality is a growing concern on the Semantic Web. The amount of data available is growing faster than ever, and the emphasis thus far has been on creating and interlinking data without much regard to how good the data actually is. The trend is shifting from creating new data to refining what already exists.
Data quality is a subjective concept and a formal representation for it is often troublesome. First, we must define what is meant by data quality - what are the different facets of the concept. Second, a way for representing this quality must be found. Third, actual processes to refine data and improve its quality and ways to take data quality into account on the Semantic Web must be developed.
This work presents some solutions to the problem. Many ways to annotate quality metadata as RDF are first discovered, along with their pros and cons. A framework for managing RDF-based quality metadata is presented, with a set of tools for specifically managing the quality annotations. Additionally, an automatic annotation system and a schema validation system, within the restraints of the open world assumption, have been designed, implemented and integrated into the framework. The system has been tested using real life datasets with promising first results.

Keywords: semantic web, ontologies, data creation, validation, data quality

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Joonas Laitio

Työn nimi: Semantic Web Data Quality Control

Päivämäärä: 26.09.2011 Kieli: Englanti Sivumäärä:6+63

Automaatio- ja systeemitekniikan koulutusohjelma
Mediatekniikan laitos

Professuuri: T-75

Valvoja ja ohjaaja: Prof. Eero Hyvönen

Datan laatu on kasvava ongelma semanttisessa webissä. Saatavilla olevan datan määrä kasvaa nopeammin kuin koskaan, ja pääpaino on tähän asti ollut datan luonnissa ja yhdistelyssä sen laadun sijaan. Nyt huomio on siirtymässä uuden datan luonnista olemassaolevan datan laadun jalostamiseen.

Datan laatu on subjektiivinen käsite, ja sen formaali esittäminen on usein mutkikasta. Ensiksi täytyy määritellä se, mitä tarkoitetaan datan laadulla, ja mitkä ovat käsitteen eri puolet. Tämän lisäksi täytyy löytää sopiva tapa laadun esittämiseen. Lopulta tulee myös kehittää varsinaisia prosesseja datan jalostamiseen ja laadun parantamiseen, ja tapoja ottaa tämä laatutieto huomioon semanttisessa webissä.

Tämä työ esittää joitain ratkaisuja näihin ongelmiin. Monia tapoja merkitä laatutietoa on esitetty, hyvine ja huonoine puolineen. On kehitetty järjestelmä RDF-pohjaisen laatutiedon hallintaan, ja joukko työkaluja jotka on räätäilöity tämän tiedon hyödyntämiseen. Lisäksi on kehitetty yleinen automaattisen annotaation rajapinta ja skeemavalidaatiojärjestelmä avoimen maailman oletuksen asettamien rajoitusten puitteissa. Nämä on toteutettu ja integroitu yleisempään hallintajärjestelmään. Järjestelmää on testattu käyttäen reaalimaailman käyttötapauksia ja aineistoja ja ensimmäiset tulokset ovat lupaavia.

Avainsanat: semanttinen web, ontologiat, datan luonti, validaatio, datan laatu

Foreword

The author wants to give his thanks to the supervisor and instructor of this work, Eero Hyvönen, for his effective and readily given guidance in the making of this work, as well as the work and research that eventually led to the thesis. Thanks also belong to the other members of the Semantic Computing Research Group, based in the Department of Media Technology of the Aalto University School of Science.

The writing of the thesis and most of the software development included has been a part of the National Semantic Web Ontology project in Finland (FinnONTO, 2003 - 2012), funded mainly by the National Technology and Innovation Agency (Tekes) and a consortium of 38 organizations.

Otaniemi, September 26, 2011

Joonas Laitio

Contents

1 Introduction

The future of knowledge and data is going towards the semantic web with every passing year. The trend is clear: the amount of data on the Internet grows exponentially, but actually getting relevant data as an end user is challenging. This challenge stems from the basic structure of the Internet. In its core it's simply a loosely interlinked *"web of pages"*, the contents of which is purely made to be read by humans. Aggregating and centralized representation of data is very hard without machine readable data about the contents. Semantic web offers a solution to this problem.

The aim of the semantic web is to extend the web of pages and strive towards a web where the semantic meanings of the data are machine readable in addition to being human readable. This way the actual semantic *contents* of the pages would be interlinked to form a *"web of data"*, in addition to the pages being interlinked, as illustrated in Fig. 1. A machine readable data format and natural text must be used in parallel, so that the semantic bits and pieces that the page is made of can be linked to other bits and pieces on other pages. The most important of these machine readable formats is Resource Description Framework, RDF, which is described in more detail in Sec. 2.1.

To enable people to adapt the semantic web into actual use, the creation and management of the data used in it must be as easy and effortless as possible. Because data conversions from one format to another are typically very error-prone operations, there must exist robust and effective tools for ensuring the quality of the result data. Depending on the use case, the importance of the quality of the data varies greatly. This means that the tools must not only be very versatile, but also very simple for use cases. The Open World Assumption, described in more detail in Sec. 2.5, emphasizes further the need for quality control; in the open world semantically erroneous data can cause reasoning errors that are very hard to track to their source.

Research about data quality on the Semantic Web has not focused on quality control on a practical level. There are some extensive and formally robust frameworks for representing imperfection, especially when working with ontologies [2–4], but little to no experiences from actually applying these to real use cases. The need for quality control is even greater when working with automatic or semi-automatic annotation tools than manually with ontologies. Looking for and correcting errors in machine-created instance data is a very time consuming process in the absence of proper tools. It would be easier if the data creation process itself would take quality control into account.

The three main research questions studied in this thesis are the following: how can data imperfection be taken into account on the semantic web, how can different types of imperfect data in RDF format be annotated with quality metadata, and how can data quality control be utilized in automatic or manual metadata creation. Below the questions are described in more detail.

How can data imperfection be taken into account on the semantic web? It is clear that data quality is in the heart of it all: if we want to enable machines to

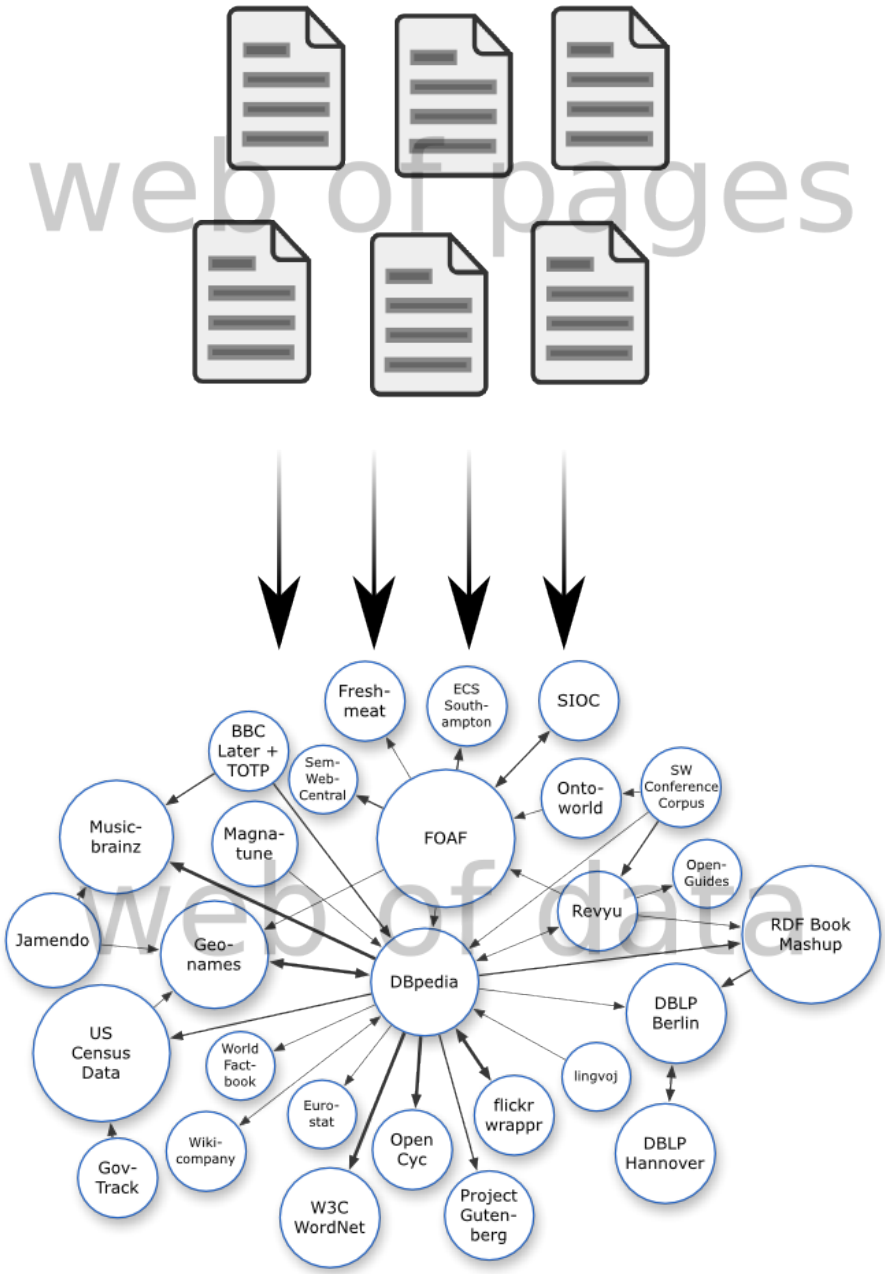

Figure 1: The transformation from a web of pages to a web of data [1]

reason non-trivial semantic conclusions, we want the basis for that reasoning to be as robust as possible. However the ways in which this needs to be taken into account are not simple. This thesis addresses the question when the data quality related things must be emphasized and how they should be handled.

How can data imperfection be annotated as metadata for RDF-based datasets? What this practically boils down into is creating additional triple-specific annotations, such as confidence values. Adding such a fourth value into a triple is a tricky

problem. There is a multitude of solutions for this problem, each with their own pros and cons. This thesis reviews the best and most used solutions and applies one of them to solve the matter in practice.

How can data quality control be utilized in automatic or manual annotation? Clearly there are theoretical justifications for adopting quality control, but the wall between theory and practice must be overcome as well. In the design and implementation section (Sec. 5) a prototype system for metadata quality control is presented.

2 Semantic Web and Linked Data

This section covers the basic technical theory of the Semantic Web, beginning from the languages and data representations used and the context of the most important constructs used in them. Later the basics of what quality means on the Semantic Web and how quality control could be approached are discussed.

2.1 RDF

RDF (Resource Description Framework) [5] is a data format where data consists of *resources*, i.e., specific entities with identity such as objects, concepts or people, and *literals*, i.e., words and numbers with no other intrinsic meaning in addition to their syntactic form. Resources are represented by globally unique identifiers, URIs, so an abstract entity can be encapsulated as a piece of data. By convention, URIs themselves are strings similar in appearance to web page addresses, URLs, though the two should not be confused with each other.

Resources are linked to literals and other resources with *statements* represented as triples. Triples are simple three-part constructs that have a subject, a predicate and an object. Each triple in RDF is a statement saying that the subject has a certain *property* (predicate) with a certain value (object). A single triple is the smallest unit of information that can be expressed in RDF. A set of triples that interlink resources with each other forms an *RDF graph*. [5]

Fig. 2 depicts an exemplary RDF graph with five triples. They connect a total of four resources and two literals (by convention represented by ellipses and rectangles, respectively) to each other. The triples ultimately state that "A person called Joonas Laitio goes to a school called Aalto University".

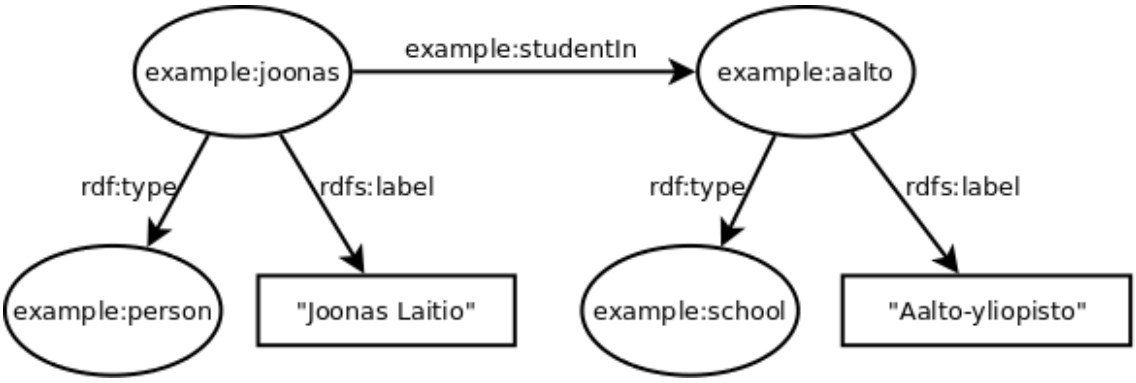

Figure 2: An example RDF graph

In the above graph, the URIs for the resources are written in their shortened or *prefixed* form for brevity. For example, in its full form the URI `example:joonas` would in this case be `http://www.seco.tkk.fi/example/joonas`. The *prefix mapping* between these short and long forms is always supplied along with the graph. [5]

2.2 RDFS and OWL

RDF is the basic data format. RDFS (RDF Schema) [5] and OWL (Web Ontology Language) [6] are definitions for a basic set of ground rules for using that data format to store semantically sound data. RDFS provides the basic language constructs for describing vocabularies, and OWL provides richer ways to describe whole ontologies. [6]

This section covers the most important basic features of these languages in the context of this work. Since the subject matter of this work deals closely with the semantics of most of these RDFS and OWL constructs, it is necessary to first study them in detail in order to build upon them when making quality assertions about the data.

2.2.1 Classes and Instances

A class is a generalized description of a set of individuals, which are in turn called instances of that class. For example in the graph in Fig. 2 the individual named "Joonas Laitio" is represented by the URI `example:joonas` identifying an instance of the class person, represented by the URI `example:person`. Another way to say this is that instances have a *"is-a"* relation with their class: Joonas Laitio is a person. [7]

Classes and instances are said to be on different meta levels. When talking about classes we are not talking about the specific instances, but rather abstract generalizations about them. There are also *meta-classes* that are a meta level above classes: for example the concept of *class* itself is a meta class , because it is the the class of regular classes such as *person*. Instance to class relations are defined by the RDF property `rdf:type`, as shown in Fig. 3. *Type* is also often used as a synonym for *class* when dealing with RDF data. [8]

2.2.2 Domain and Range

Domain and range are the fundamental properties that state where and with what values various properties are used. Formally, a triple `(x rdfs:domain y)` states that resources that have a property `x` belong to class `y`. Similarly, a triple `(x rdfs:range y)` states that resources that are used as values for the property `x` belong to class `y`. For example, a property "has published" could have "person" as domain and "article" as range, because in general persons publish articles.

Domain and range definitions allow for powerful reasoning semantics. If we have the above definitions in our knowledge base and come across any resource that uses the "has published" property, we can reason that the resource is a person and that all the values on that property are articles, even if we don't have explicit data that states this. Great care must therefore be taken to use properties in their intended context: otherwise reasoning based on domain and range definitions can easily provide erroneous and completely useless data. Later sections discuss the usage and context of domain and range definitions further.

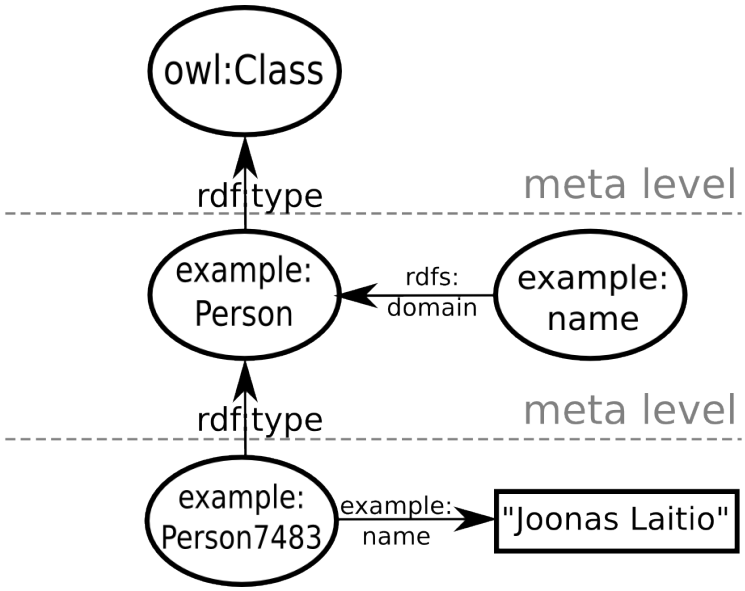

Figure 3: Illustration about meta levels in OWL

2.2.3 Restrictions

OWL restrictions provide a way to express other constraints on the values of properties than just their class. Perhaps the most common restrictions are cardinality restrictions: they restrict the number of values that a certain property on an instance of a specific class can have. For example, the cardinality of the property `parent` on class `person` is two. The other kind of OWL restrictions constrain all or some of the values on the property to a specific set of resources.

An OWL restriction is actually a nameless class that represents all entities that conform to its constraints. A restriction that states that the cardinality of the property `parent` is two actually means that all entities that have two values on the `parent` property belong to that class. The restriction class is tied to another class, such as `person`, through a *subclass of* relation. We must define `person` as being the subclass of our cardinality restriction for the restriction to apply to that class. The *subclass of* relation effectively states that `person` is one kind of entity that has exactly two values on its `parent` properties.

2.3 RDF Serializations

RDF is just an abstract way to represent data in the form of a graph that consists of triples. There are several ways to serialize that data to be stored in a concrete form as a data file. This is a distinct difference from many other data formats, most notably XML, for which the abstract data format and concrete data serialization are one and the same. Below, the most common types of RDF serialization formats are listed, along with an example serialization of the RDF graph in Fig. 2.

RDF/XML

XML [9], due to its extensibility, is one of the most widespread formats for storing data as textual files. This also holds true for RDF: the XML representation of RDF, called RDF/XML [10], is the W3C recommendation for serializing RDF data. However, the format is very verbose and not very human-readable; furthermore, most XML-specific technologies such as XPath [11] cannot be utilized very well for it and it is losing popularity.[12] It even slightly restricts the kind of RDF that can be represented by it: some predicate URIs and certain Unicode code points cannot be used.[13] A simpler and more usable XML serialization for RDF called TriX has also been developed [14], but it is not widely used since it is not a W3C recommendation.

An RDF/XML serialization begins with namespace declarations as attributes of the root element. Then, the subject resources are listed as their own `rdf:Description` elements, with URI references listed as `rdf:resource` attributes and literals as simple text, with the element name being the URI reference to the property of the triple. There are a lot of repetitive segments and the URIs are not always shortened with their respective prefix.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:example="http://www.seco.tkk.fi/example/" >
  <rdf:Description rdf:about="http://www.seco.tkk.fi/example/joonas">
    <example:student_in rdf:resource="http://www.seco.tkk.fi/example/aalto"/>
    <rdf:type rdf:resource="http://www.seco.tkk.fi/example/person"/>
    <rdfs:label>Joonas Laitio</rdfs:label>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.seco.tkk.fi/example/aalto">
    <rdf:type rdf:resource="http://www.seco.tkk.fi/example/school"/>
    <rdfs:label>Aalto-yliopisto</rdfs:label>
  </rdf:Description>
</rdf:RDF>
```

Turtle (N3)

Notation 3 [13] is a versatile data format that can be used to store different kinds of data. A subset of it, Turtle, is sufficient for storing triple based data such as RDF.[13] It is much more compact and human-readable as RDF/XML. The prefix definitions are located at the start of the serialization and the triples are listed by subject after that, similar to RDF/XML, but abandoning the constraints of XML enables the format to use much less repetition to describe the same thing. Since *rdf:type* is a very common property, it is given an alternative representation, a simple `a`.

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix example:  <http://www.seco.tkk.fi/example/> .

example:joonas
      a       example:person ;
      rdfs:label "Joonas Laitio" ;
      example:student_in example:aalto .
```

```
example:aalto
      a       example:school ; rdfs:label "Aalto-yliopisto" .
```

N-Triples

N-Triples is in turn a subset of Turtle, and subsequently N3. It's purpose is to be an extremely simple RDF serialization format. It does not store URI prefixes or resources as complete entities—instead, it simply stores triples one by one, with each row consisting of the subject, predicate and object of the triple. The format is not very human-readable, but it is extremely simple to produce and fast to parse with computers. Thus, there are clearly defined use cases where using it is beneficial: when the data needs to be efficiently streamed either as input or output.

```
<http://www.seco.tkk.fi/example/aalto> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.seco.tkk.fi/example/school> .
<http://www.seco.tkk.fi/example/aalto> <http://www.w3.org/2000/01/rdf-schema#label> "Aalto-yliopisto" .
<http://www.seco.tkk.fi/example/joonas> <http://www.seco.tkk.fi/example/student_in> <http://www.seco.tkk.fi/example/aalto> .
<http://www.seco.tkk.fi/example/joonas> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.seco.tkk.fi/example/person> .
<http://www.seco.tkk.fi/example/joonas> <http://www.w3.org/2000/01/rdf-schema#label> "Joonas Laitio" .
```

2.4 Semantic Web

The semantic web can be described as a collection of technologies that aim for representing semantic meanings in a language that computers can understand. If machines can understand the very basics behind human reasoning processes, they can be used to represent and solve complex problems intuitively, similar to how humans approach them. Of course, the realization of this goal is extremely complicated—but nevertheless a task worth pursuing.

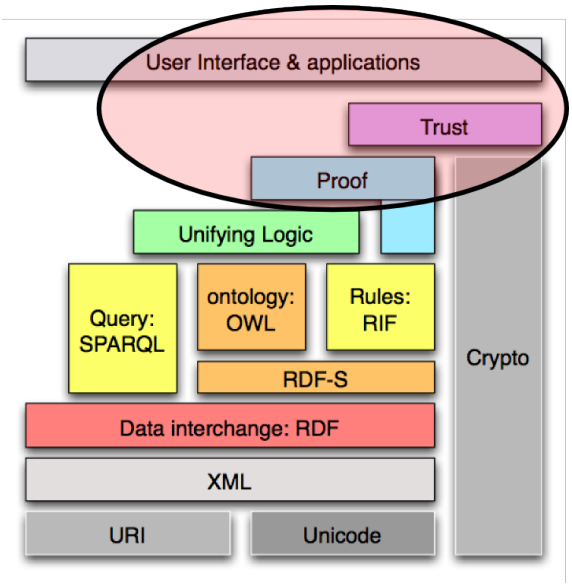

Figure 4: Semantic web layer cake

The technologies used for the semantic web are usually represented by a stack of layers: each layer depends on the layers below it. The structure of this "semantic web layer cake", one version which is shown in Fig. 4, is a subject of ongoing debate.[15] Nevertheless, the basic premises remain the same: everything is built upon the basic web markup conventions (Unicode, URI) and data formats (XML, RDF). On this the foundations for machine intelligence are built: semantic web logic, rule and query languages and technologies (RDFS, OWL, RIF, SPARQL), applied data subjectiveness (proof, trust) and finally the user interfaces and the ways for a human to actually dive into the world of intelligent data. Throughout the stack is the regard for privacy and security: data encryption and other cryptographic measures concern many different levels on the stack.

The main focus of this thesis is the circled section of Fig. 4: the Proof and Trust layers along with some user interface and application considerations. This is the area where the ways in which data subjectiveness can be annotated and otherwise taken into account are of special interest.

2.4.1 Proof Layer

"The Proof layer involves the actual deductive process as well as the representation of proofs in Web languages (from lower levels) and proof validation." [15]

The Proof layer has to do with actually deducing things: answering semantically phrased questions about the nature of things in a general way. An example would be granting access to a restricted web site: in this case the proof would be a collection of data and reasoning rules from which it can be deduced that the user is allowed access. [16, 17]

2.4.2 Trust Layer

"The Trust layer will emerge through the use of digital signatures and other kinds of knowledge, based on recommendations by trusted agents or on rating and certification agencies and consumer bodies. Sometimes 'Web of Trust' is used to indicate that trust will be organized in the same distributed and chaotic way as the WWW itself" [15]

The trust layer directly deals with data quality control, and subsequently lies in the heart of this work. It should provide the *means* to get the confidence data, through a possibly complex process of evaluating the certainty of some piece of knowledge with all the other resources at our disposal. The resulting trust knowledge is either used in real time or stored as *quality annotations*, discussed later on.

2.5 Open World Assumption

A major feature of semantic web data is its distributedness. As knowledge on the web is typically fragmented into small pieces located in multiple places, we must begin from the assumption that the data at hand at any single given time is incomplete. Hence we can't assume that the data we have is erroneous, since there might be another piece of data that explains an apparent conflict.

This principle is called the Open World Assumption, as we consider the world around our data to be open and never complete and self-sufficient. For example, consider the situation where some property, such as "has father", is constrained to have only one value. If we have two different resources as values for this property, we might just be missing the information that these two resources in fact describe the *same entity* and are one and the same.

2.6 Linked Open Data

One of the hottest focal points in contemporary semantic research is linked open data.[18] It was known from the start that the semantic web needs vast amounts of data to produce interesting results by machine reasoning: linked open data is the answer. Basically, linked open data means metadata that is open for everyone on the Internet and what machines can automatically go through and evaluate. The *linked* portion means that the datasets in different stores are linked to each other semantically; this is achieved through the use of URIs to denote resources.

The amount of linked open data on the Internet is growing at an astounding pace, as are the links established between different sets of data. Fig. 5 shows a representation of the "linked open data cloud" [1], a figure showing the most important linked open datasets and how they interlink with each other. The figure is an exemplary one from July 2009; since then, the amount of datasets in later versions of the diagram has over doubled [1] and the cloud is too large to show here. The data breakthrough is clearly here—now it's time to ascertain that the quality of the data is sufficient for versatile use.

2.7 Quality in the Open and Closed World

In the open world it is hard to measure data quality explicitly. The whole concept of high quality data in the open world is somewhat fuzzy. As we always start with the assumption that the amount of data we currently have is insufficient, it is practically impossible to say anything about the quality of any data in a strict sense. Everything is possible unless stated otherwise.[19] Though this approach has good reasons for it and it is suited very well for a distributed web of data, it makes data validation problematic.

The opposite situation is the closed world assumption, which is more typical and traditional in most data applications dealing with bounded data. Every piece of data needs to have a specific place to contain it, such as a specific column in a database. Things we do not know anything about, or information we don't have a place to put in, can be assumed to be false. This makes for very easy validation. We can go through every piece of data and validate it against the specific constraints that belong to it. Violations of these constraints are simply errors.

In a way, validating data that typically resides in the open world, such as RDF data in the semantic web, requires "closing the world" around some specific piece of data. If we do this, we can interpret schema definitions in a strict sense and dictate what is *allowed* instead of what is *possible*. This approach is discussed in

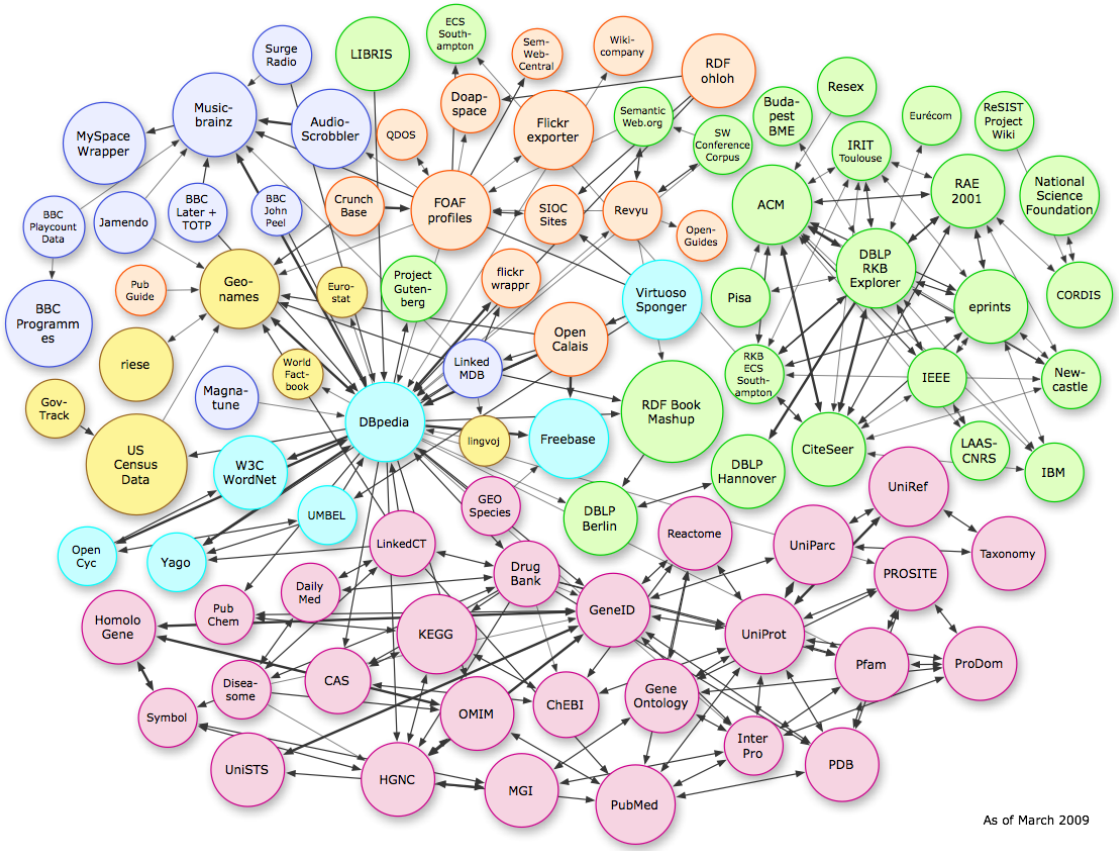

Figure 5: Linked Open Data cloud diagram [1]

more detail in Sec. 4.1. Data is divided into three subtypes, and imperfect data is further divided into many types of imperfection.

2.8 Types of Imperfection

The different types of imperfect data must be understood and enumerated so that comprehensive quality control is possible. These types have often only minute semantic differences and are partly up for definition—for the sake of consistency the English terms for the types, along with their Finnish counterparts in parenthesis, are listed here as they are used in the context of this work. A diagram of the types can be seen in Fig. 6

Valid (käypä, validi)

> Valid data is data without imperfections—no evidence has been found of the features of imperfect data listed below.

Imperfect (epätäydellinen)

> Imperfection is used as a superordinate term for all kinds of minor or major unwanted aspects of the data. These can be unwanted in general or in the

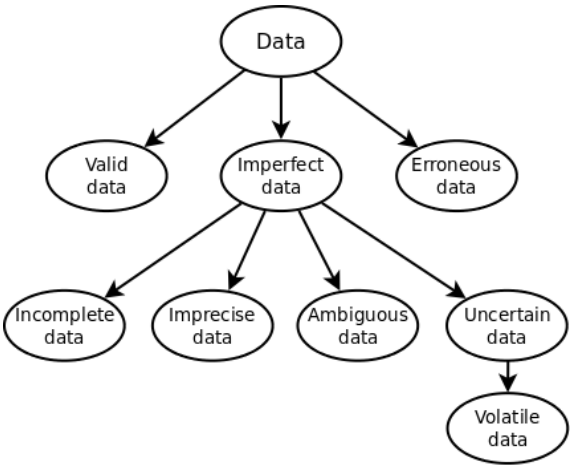

Figure 6: Types of data by quality

context of a certain application. Unlike erroneous data, imperfect data has the makings of valid data but fail in some aspect or aspects.

Erroneous (virheellinen)

Erroneous data is known to be false. Even this kind of data should not necessarily be deleted immediately – even erroneous data can be useful in some applications. For example, one might want to explicitly mark erroneous all common misconceptions about some subject. Unlike imperfect data which can be fixed to be valid with relatively minor changes, fixing erroneous data requires a drastic alteration to the underlying semantics of what the data means.

Imperfect data is further divided into subordinate terms, listed below.

Incomplete (puutteellinen)

Incomplete data means that the data at hand is not enough for satisfactory definition of something, even though all the data is fundamentally valid. For example, we might have fragments of some historical text, but not a complete version. Thus, our knowledge of the text is incomplete. Incomplete data is a concept that, like many things in RDF data validation, clashes with the Open World Assumption (Sec. 2.5).

Imprecise (epätarkka)

Imprecise data means insufficient data accuracy level or excessive granularity. We might know the amount of participants in a historical battle only to be in the range 50 000 – 500 000, which would be too imprecise for an application that demands the amount with an accuracy of 10 000 people. Impreciseness is quite application dependent and usually needs a context in which to assess it—data that is too imprecise for some uses might be perfectly fine for others.

Ambiguous (epäselvä, monitulkintainen)

Data is classified as ambiguous when there are multiple interpretations about the state of something, possibly from multiple sources. For example, different historians may have given different estimates on the amount of soldiers that participated in a historical battle. In this case all the options must be marked as ambiguous if there is no way to assess one of the sources as more reliable than the others. Perhaps the most common case of ambiguity arises when matching natural language to a vocabulary or ontology: lemmatizing (removing inflection of words to acquire the base form) is by nature an ambiguous process because of the existence of *homographs*, words that share the same spelling but have different meanings.

Uncertain (epävarma)

Data is uncertain when there is reason to doubt its truthfulness. Uncertain data is further divided into objective and subjective uncertainty; objective uncertainty is caused by things such as malfunctioning measuring devices, unreliable human sources and uncertainty about if a historical event really happened or not. Subjective uncertainty concerns concepts that are difficult to objectively categorize, such as knowledge whether or not a person is "old" or not. Uncertain data can be paired with confidence values telling the probability with which the data is valid.

One specific subordinate of uncertainty is *volatility*. We might know for sure that the data is valid upon creation, but if we know the data is subject to rapid change it is nevertheless imperfect. Weather phenomena such as wind speed and direction are examples of volatile data.

What is discussed above is about the absolute quality of the data, indifferent of the application it is used for. However, the quality of data is often relative to the context in which it is used, and even more so in the future.[20] Generally speaking, there should be no obstacle why this kind of *contextual quality* couldn't be controlled with the same methods as absolute quality.

Contextual quality can also be divided into different subtypes in a similar way as absolute quality. Perhaps the most prevalent of these types is *irrelevant* data. Often an application or task is focused around a specialized purpose, and much of the overall data at hand is irrelevant. Optimally, all the irrelevant data is filtered out based on a conception of contextual quality at that time. [21]

3 Related Work

This thesis presents a framework where good quality semantically rich high quality data can be produced. There is naturally a lot of related work in the area of enriching text documents or textual property values with semantic concepts and other links to external resources via using automatic annotation and similar techniques. Most of these do not think of quality control as a separate entity or feature and simply use built-in machine learning algorithms for ranking data. These systems generally do not give confidence values or similar quality metadata as output.

A well known semantic enrichment system is OpenCalais[1] [22]. In the core of OpenCalais is a web service: unstructured data, such as a text document goes in, and out comes a semantically enriched version of it; interlinked with resources such as people, organizations, events and facts that are related to the contents of the document. It is mainly domain independent and utilized for a wide range of subjects [22–24]. Another similar system is Zemanta[2], providing much of the same functionality. It is also provided both as browser extensions, so the users can semantically enrich any text they input in a browser, and server side plugins for common content management systems such as Wordpress and Drupal.

These systems are good at what they are meant for [25], but not as general solutions to the problem in this thesis. They are presented as black-box knowledge engines: they only work for a limited set of languages, and even though they are mostly domain independent, they work much better for some domains than others. Most importantly, in the context of this work, the quality is in the hands of the system, not the user. No confidence or similar information is provided.

For publishing one's open datasets and providing a distributed collaboration environment to modify them, the most common approach is to use CKAN (Comprehensive Knowledge Archive Network)[3]. CKAN provides a way to publish and edit structured data, while maintaining a searchable index for the data stored within. Pertaining to the Semantic Web, the open datasets of CKAN are also available in RDF format[4]. However, this does not bring any more semantics into the data. As CKAN uses simple tagging instead of concept ontologies, the data relations inherit many of the problems the Semantic Web is trying to solve such as ambiguous or unclear tags and lack of multilingual support. There are a number of commercial alternatives to CKAN that also suffer from this lack of semantics, such as Infochimps[5] and Socrata[6].

A general approach to data-driven RDF applications is provided by the Callimachus Project[7], a framework for easy building of RDF management applications, among other things. A system similar to the one described in this thesis could, for the most part, be built using Callimachus, and in general the all-encompassing

[1] http://www.opencalais.com/
[2] http://www.zemanta.com/
[3] http://ckan.net/
[4] http://semantic.ckan.net/
[5] http://www.infochimps.com/
[6] http://www.socrata.com/
[7] http://callimachusproject.org/

paradigm is interesting in the context of data quality control on different levels. However, the aim of this thesis is to focus attention to specific aspects of metadata control with specifically tailored functionality to suit these needs.

Closer to the level of abstraction that this thesis will present, there are some good tools for distributed collaboration on actual RDF data, triple by triple. The most prevalent of these are PoolParty[8], focusing on SKOS Thesaurus Management [26], and Web Protégé[9], focusing on ontology development [27]. They are somewhat streamlined, perhaps lacking in extensibility, but they excel at their intended use area.

Version control is a concept close to that of explicit quality control. OntoView [28] is a versioning system for ontologies, which has the basic features of version control: storing each version of a developed ontology and providing difference information between any two versions of it.

In the other end of the spectrum, we have work that deals not with the creation of new data, but only validating it, against a schema or otherwise. Much of this work concerns XML, and due to the slight overlap between XML and the RDF world, also concerns RDF. However, simple DTD or XML Schema definitions are not very useful for validating XML-serialized RDF due to the inherent disparity between XML and RDF and the numerous and sometimes unintuitive ways RDF graphs are expressed in XML.

XCSL and Schematron are among the more expressive XML validation and constraint languages, and have functionality that is also useful when working with RDF [29, 30]. Since they're not actually meant for graph-based data models or RDF in general, they do not work as general solutions, but can serve as baselines for other systems. For RDF data using RDFS/OWL, there are some validation systems that use very simple rules for checking common problems such as illegal URIs, such as Jena Eyeball[10]

One well-rounded language with which constraint-like characteristics of a dataset can be defined is Nepomuk Representational Language, NRL. NEPOMUK[11], a framework for Social Semantic Desktop applications, needed a generalized way to store and validate its own metadata, and the solution was NRL. It serves its function well, but nevertheless needs to be specifically adopted when building one's own schemas, and doesn't provide a way to explicitly store confidence information. [31, 32]

Developing ontologies has been a big part of the growth of the Semantic Web, and many tools and methodologies for ensuring consistency on this higher, ontological semantic level have been developed. This is the other way to approach validity: not from the syntactic level, bottom-up, but rather from the internal logic implications, or top-down. The top-down approach includes pure logical methodologies such as OntoClean [33] to remove redundancy and otherwise ensure ontological quality, and pure development and evaluation tools such as OntoWiki [34] and ODEval [35].

[8] http://poolparty.punkt.at/

[9] http://protegewiki.stanford.edu/wiki/WebProtege

[10] http://openjena.org/Eyeball/

[11] http://nepomuk.semanticdesktop.org/

From a quality perspective, the focus of this thesis lies between higher semantic evaluation, like that of OntoWiki and ODEval, and the pure syntactic evaluation level that XML Schema, or the W3C RDF Validator[12] provides. From a data creation perspective, the aim is to provide a framework description for a system with more fine-grained control on the created data than OpenCalais or Zemanta provide, but with some automatic semantic enrichment and quality-focused abstractions, which are not provided by the basic distributed RDF collaboration tools such as PoolParty or WebProtege.

[12]http://www.w3.org/RDF/Validator/

4 Methods and Markup

This section discusses actual methods for data quality assessment and control, both on a theoretical and practical level. The methods include basic approaches to the whole concept of quality on the Semantic Web, as well as concrete ways to actually annotate quality metadata in RDF form.

4.1 Schema-based Validation

Schema-based validation is usually the first thing that comes to mind when talking about the quality control and validation of big datasets. Especially data stores such as most XML formats and relational databases rely heavily on schema validation – the latter are completely built on it. Intuitively one might think that the same principle can effortlessly be carried over onto the semantic web and RDF data; after all, XML is even one of the main serialization forms for RDF graphs. However, the matter is not as simple as that.

The languages that are used for describing the structure of an RDF model, RDFS (RDF Schema) and OWL (Web Ontology Language), are not schema languages in the traditional sense. Because the Open World Assumption is generally in effect in RDF models, actual data validity can not be reasoned through the RDFS or OWL descriptions of a dataset. The schema is used for reasoning *additional* information, instead of validating the existing information. Let's assume our schema tells that "The members of the Virtanen family are policemen" (more formally, it is not false to list their occupation as "policeman", i.e., at least some of their occupation values are that), and our data contains info that tells that "The father in the Virtanen family is a fireman". This can be expressed as an RDF Turtle (Notation 3) [13] serialization as follows:

```
example:Occupation
rdf:type owl:Class .

example:Policeman
rdfs:subClassOf example:Occupation .

example:Fireman
rdfs:subClassOf example:Occupation .

example:Virtanen
owl:Restriction [
owl:onProperty example:hasOccupation ;
owl:someValuesFrom example:Policeman
] .

example:FatherVirtanen
example:hasOccupation example:Fireman ;
rdf:type example:Virtanen .
```

By Closed World standards our data is erroneous since it does not conform to the schema definition: It states that some of every Virtanen's occupation values must be *Policeman*, i.e., every Virtanen must have *Policeman* listed as their occupation, but Father Virtanen only has *Fireman* listed. The Open World Assumption states that our data is not erroneous and we can in fact make (automatic) additional reasoning: either Mr. Virtanen is *both* a fireman and a policeman at the same time:

```
example:FatherVirtanen example:occupation example:Policeman .
```

or that a fireman, in the context in which our concept *example:Fireman* is defined, also holds the semantic meaning of what it means to have the conceptual occupation of *example:Policeman*, for example in areas where the same individual handles these tasks:

```
example:Fireman rdfs:subClassOf example:Policeman .
```

or that the concepts *example:Fireman* and *example:Policeman* are semantically equivalent types of occupation (though this can be refuted with common sense reasoning: clearly they are not):

```
example:Fireman owl:equivalentClass example:Policeman .
```

This kind of thinking is well suited especially for distributed environments such as the Internet [36], where we can seldom assume that all relevant data is at hand. The downside is that the same thinking is very ill-suited for applications where clear knowledge about the validity of a certain set of data.

This desire to get validation information about a certain dataset does indeed arise on the semantic web as well. It is especially needed in situations where we want to analyze the complete dataset that some reasoning application uses. Extensive semantic reasoning can lead to large problems if it is done on erroneous premises. An example of such a system is the Finnish culture portal CultureSampo – the portal utilizes converted data from dozens of sources, the melding of which could not be guaranteed to be error free.[37] In a situation such as this we must forget the Open World Assumption and "close" the world around our desired dataset for validation purposes. Let us consider the definition of the property *place of discovery* of a museum artifact in CultureSampo (as per the actual schema used in the system):

```
kulsa-schema:place_of_discovery
    rdf:type owl:ObjectProperty ;
    rdfs:domain kulsa-schema:museum_item ;
    rdfs:label "place of discovery"@en , "löytöpaikka"@fi ;
    rdfs:range kulsa-place:places ;
rdfs:subPropertyOf kulsa-schema:place_property .
```

If someone accidentally uses this property in the place of the property *place of collection*

```
kulsa-place:place_of_collection
      rdf:type owl:ObjectProperty ;
      rdfs:domain
              [ owl:unionOf (kulsa-schema:musical_work kulsa-schema:poem)
              ] ;
      rdfs:label "keräyspaikka"@fi , "place of collection"@en ,
        "plats för uppsamling"@sv ;
      rdfs:range kulsa-place:municipality ;
      rdfs:subPropertyOf kulsa-schema:material_place .
```

to denote the collection place of a folkloric poem, that poem would be reasoned to also be a museum artifact in addition to being a poem: a clear reasoning mistake. If we close the world around the dataset for the validation (a similar assumption to what is done in the Nepomuk Representational Language [32]), we can interpret the domain clause strictly and display a warning if a property is used outside its immediate domain. A prototype system called VERA (see Sec. 5.7) was tested when the data of CultureSampo was being finalized and the preliminary results were encouraging. [38]

In conclusion, schema validation can be done on the semantic web as well, but it's not a general solution. Using it is always case specific, and it can't feasibly be automatically applied for new datasets solely based on the schema and data that supposedly conforms to it. However it is the only way to gather implicit validation information about the data, and thus it has its place in the quality control of semantic web datasets.

4.2 Quality Annotations

Schema-based validation and similar methods have some problems that have to do with its conflicts with the Open World Assumption based thinking. A semantically lighter way that doesn't conflict with it in the same way is a model, which this work calls *quality annotations*. The basic principle of it is that the data contains explicit info of what we know about the validity of individual pieces of data, or triples in the case of RDF.

The main problem of this approach is that RDF and the usual specifications that are used with it contain no standardized way to annotate data quality. The research that has been done so far also has not focused on data quality control very much at all, and thus there are no widespread conventions or pseudo-standards to tackle this problem. In this section some previously suggested quality annotation models are studied, with some new possible solutions as well, along with the pros and cons of each choice.

There are some proposed extensions to standard RDF that aim for bringing relational database features, like strict datatype constraints and referential integrity to RDF data, such as the one proposed by Lausen et al.[39] However, such methods haven't acquired widespread popularity, as relational databases and RDF are wildly different paradigms and in generals solutions used in one of them are not carried

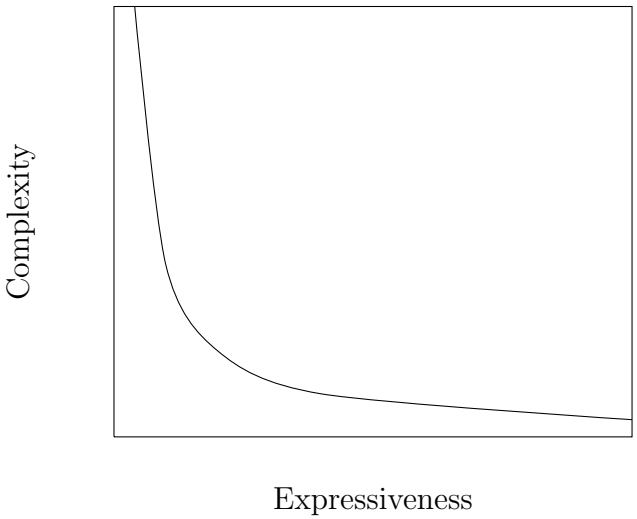

Figure 7: Trade-off between the expressiveness and complexity of a model

over very well to the other due to the confrontation between the Open and Closed World Assumptions.

One of the important defining characteristics of a data storage model is its complexity.[40] There are many differing needs for models with various extensiveness. The more extensive a model is, the more complex is the underlying model; and a complex model hinders efficient computation and reasoning. However the model must still be complex enough so that it remains versatile enough to be used in many different applications. The case is a common trade-off situation (depicted in Fig. 7) between two conflicting goals.

The premises of the semantic web rely heavily on description logic.[12, 41] With a formal logic system such as that, everything from the ground up is meticulously and axiomatically defined. Partly because of this, the same is true for all proposed systems for data quality control: they are defined from a very low level upwards, and aim for maximal representativeness at the cost of simplicity. The descriptions of these systems generally do not touch actual practice and the actual RDF representation of the system very much. [2, 3]

Then how extensive data quality control do we really need in practical applications? This question is largely unanswered in contemporary research. The research has progressed the semantic web layer cake (Fig. 4) from the ground up and actual end user applications have followed behind. Since the whole cake is meaningless without the means for the end user to consume it, it would also be useful to approach the problem from the opposite direction starting from practical use cases. This work discusses through practice what aspects of data quality control are actually useful and worth implementing.

The goal of creating good data is usually to eliminate all possible imperfections from it. Because of this, it is good to keep in mind that the finalized version of the data actually used by applications does not necessarily contain the data quality annotations. It would be beneficial if the data quality model could be separated

from the base model created with technologies such as RDFS, OWL and SKOS. However, during the creation phase the quality model and the actual data model should be integrated.

This section details various ways of quality annotation markup. For the sake of clarity, they all contain a way of marking the triple portrayed in Fig. 8 as imperfect. It should be noted though that because of the methods differ, so do the semantic implications of them. As such, all of the figures to be presented are not semantically completely equivalent to each other.

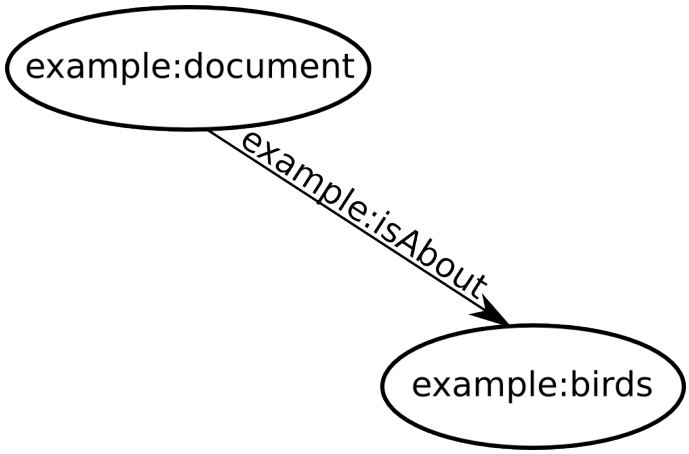

Figure 8: A basic example triple

4.2.1 Markup with a Model Expansion

Since RDFS or OWL do not as such provide ways to express imperfect data, a natural way to solve the problem is to expand them with features that allow it. One such model that is built on OWL but provides quality annotation tools is proposed by Preuveneers et al.[4] and is shown in Fig. 9. With the model one can provide direct quantitative quality control data such as correctness, confidence and resolution.

With the model one can annotate property specific confidence values directly to the schema. As a downside of this method, the literal values for different types of quality cannot be dynamically set: if we want to have a hundred different confidence values for some property, we need to create a hundred new properties. This problem stems from the triple-based structure of RDF and is present in some way in all models that do not use reification.[4] Fig. 10 illustrates how this specific model expansion can be used to mark a triple as imperfect. The original property is replaced with a custom one with a different type. Then all relevant quality information for this custom property are listed as literals. In the example, only a percentual correctness is listed, as 80%.

The major upside of using model expansion is generality – the expansion can be published with finalized formal definitions, so anyone can adopt it and easily generate data that conforms to it. The principle is widely used: many common

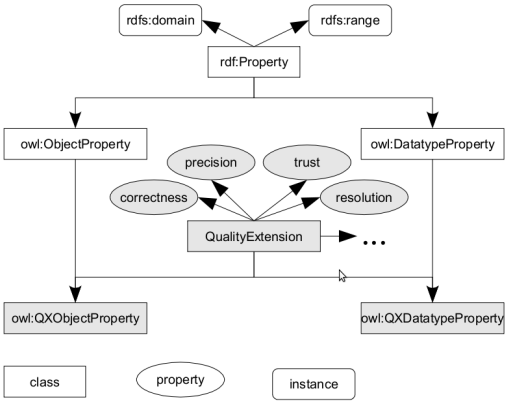

Figure 9: An OWL-extension that enables quality annotations [4]

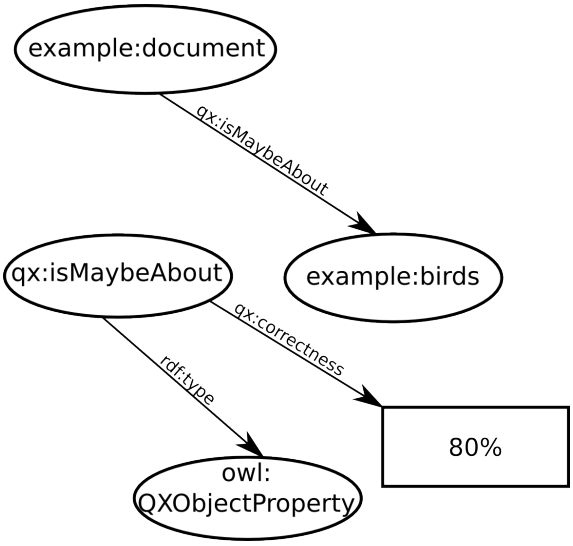

Figure 10: Data quality annotation with model expansion

vocabularies such as Dublin Core are designed with expandability in mind [42] and are expanded with dozens of different expansions [43]. However, adopting a general expansion usually leads to needless complexity, if the expansion isn't specifically designed for the application context at hand. To conform to the generally defined expansion, one might need to use parts of it that wouldn't be strictly necessary.

4.2.2 Markup with Reification

Reification is a feature in RDF that, even though it is fundamentally represented with triples, is a deviation from the basic graph/triple structure. Reification means the process of *reifying* an RDF statement (triple) by giving it an unique identifier as a whole: this enables them to be used as a part of another triple—either the subject or

the object. If used as the subject, instead of the regular `(subject - predicate - object)` structure it would be `((subject - predicate - object) - predicate - object)`. This allows native support for more complicated semantic structures such as "Weather was sunny at noon", which would otherwise be non-trivial to implement with triples.

Reification can be used for quality annotations, by marking triple-specific metadata to support or doubt the validity of them.[44]. A simple method of quality annotation markup with reification is shown in Fig. 11. Statements have been marked as uncertain by linking them to a general "uncertain resource" via an object property. This method is general in the sense that the properties can be arbitrary and there can be many of them. For example, we could easily give a numerical confidence value for the uncertainty with a literal-valued triple, similarly to Fig. 10.

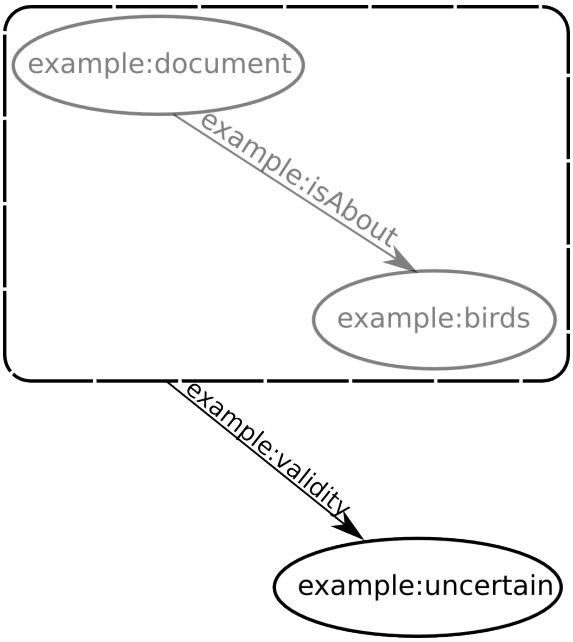

Figure 11: Data quality annotation with reification

However, reification fails as a solution for applications like this both in theory and practice. The theoretical problems have to do with the underlying formal implications of reified statements. In practice the problem is the complexity of this method: reification partly breaks the general triple structure of subject-predicate-object -statements model in RDF. This means that reification should be separately taken into account in any application that uses the data. The most common use cases for semantic web data, such as recommendation systems and automatic data enrichment, are considerably harder to implement and computationally more expensive with data that contains reified statements. [20, 45]

4.2.3 Markup with Quads

One method to achieve the wanted goal is to use named graphs for storing extra information about a triple.[46] Basically this means that instead of triples, the data store utilizes quads, where the subject-predicate-object combination additionally has a named graph (URI) attached. These kind of named graphs are directly supported in the RDF format specification [5] so there is no need to avoid straying from a basic triple structure like this. Once the triples are linked to named graphs, we can store information about the graphs: for example, their origin, method of creation and authors. This kind of extra information is usually called *provenance*, and the method is widely used on the Semantic Web by many large organizations.

The method is somewhat similar to the reification method described in Sec. 4.2.2, but is simpler and more elegant as it clutters the data less and is more intuitive and transparent. However, it is more suited to provide provenance than direct quality information since that is its original purpose. If we use it for quality information instead, like in Fig. 12, we must tag the whole graph with the same quality information, or alternatively divide the graph into a multitude of subgraphs. Having the ability to annotate single triples and resources is useful, but dividing the graph into small subgraphs is counterproductive if we also want to annotate the provenance of a graph as a whole. This is a problem as both quality and provenance information should be allowed to coexist in the same RDF dataset. An alternative is to simply track the quality of whole datasets instead of smaller units such as triples or resources [47], but that is a much more broad and general approach as the one assumed in this thesis.

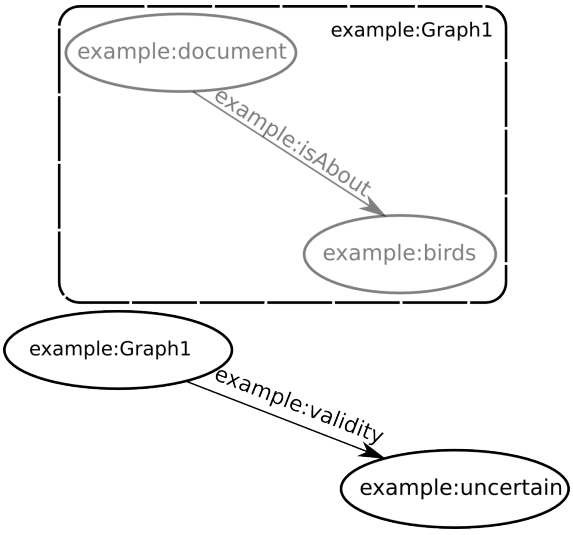

Figure 12: Quality annotation with named graphs in quads

4.2.4 Markup with Ad-hoc Local Instances

For certain applications, linking uncertain data into an existing knowledge base is enough to solve the data quality annotation problem. If we have a knowledge base we know we can trust, and some unknown data whose quality we want to annotate, it's sometimes enough to create a mapping between these datasets instead of actually figuring out confidence values and other metadata. Fig. 13 illustrates this process: we create a local instance of every possibly interesting resource, and link these to a pre-existing knowledge base using, for example, the property `owl:sameAs`[13] if we consider them to be exactly the same, or some other relation. In the case of reference ontologies and classes, like in Fig. 13 (generally *birds* instead of some specific bird entities), this relation is often `rdfs:subClassOf`[14], as it is a very strong thing to say that two classes as exactly the same. This marks them as being semantically similar and links them to the whole knowledge base, whereas local instances deemed as invalid are left to be orphans or semantic "dead ends", without links to meaningful resources.

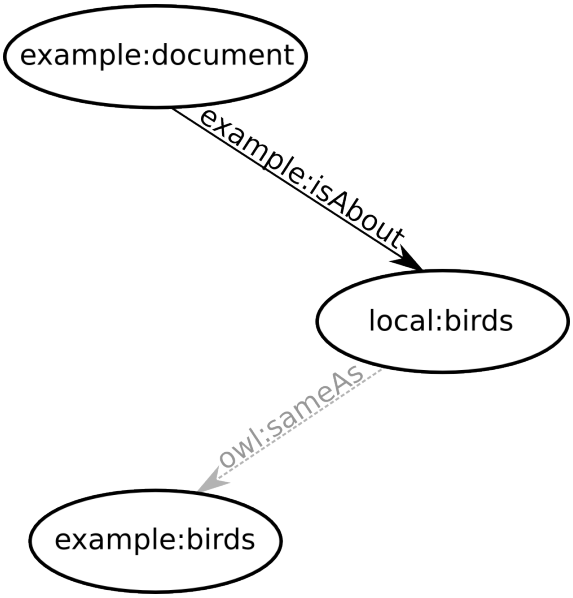

Figure 13: Data mapping using local instances

Due to the nature of this method, it has limited use. Primarily it can be used in the context of automatic annotation and converting legacy data over to the semantic web.[48] In this application, everything is about linking old data to new data, essentially the equivalence of resources. For most quality control needs, such as actually representing data imperfections, this type of model is not sufficient.

[13] http://www.w3.org/2002/07/owl#sameAs
[14] http://www.w3.org/2000/01/rdf-schema#subClassOf

4.2.5 Markup with Hierarchic Relations

The problem with the referred model expansion was the need to make explicit and numerous properties. Creating a new property for each percentage value is cumbersome and clutters the model. The problem with reification, among other things, is that it breaks the natural triple structure of RDF, even though it is intuitively a good choice. A novel way to avoid these problems, and in a way combine the mentioned two methods, is using hierarchic relations to tag triples with quality annotations. The theory behind the method is outlined here, and then applied to practice and described in further detail in Sec. 5.3.

Triples are characterized by their predicate. If the predicate is substituted with another predicate that is both a subproperty of the original one and a *general imperfect property*, it can be tagged as imperfect in a versatile and simple way, without loss of data during the process. Since the link to the original property exists through the subproperty relation, the operation can easily be reversed—the original property is an immediate superproperty.

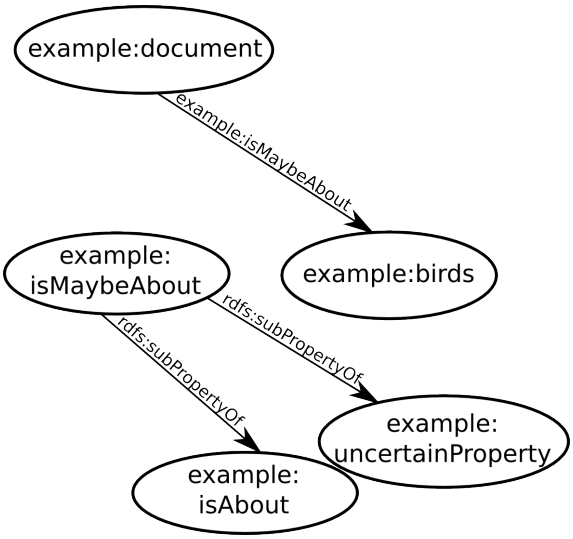

Figure 14: Quality annotation with hierarchic relations

Fig. 14 shows the example triple marked as imperfect with this type of annotation. There are many upsides with this kind of approach. The operation of marking a triple as imperfect can be reversed, i.e., the quality annotation removed so that the triple is back to its original state (Fig. 8). Because all the imperfect triples refer to the same general imperfect property, they can all be found very quickly by taking the transitive closure of that property. The method is versatile and can be applied to many varying quality annotation needs. It is also extensible, as the created properties can be given any custom properties necessary. If they were given numerical confidence values, the method is somewhat similar to the model expansion method described in Sec. 4.2.1.

One downside is that this type of quality annotation requires relatively many

triple changes when creating new properties for imperfect triples. If an existing and applicable quality annotation property is not found, a new one needs to be created and the property hierarchy altered to mark it both as a subproperty of the original property and the general imperfect property. This problem is somewhat mitigated by the fact that the properties are indeed very reusable: not every annotated triple requires a new property. As long as the substituted property is the same and the type of imperfection is the same, existing subproperties of the general imperfect property can be used.

If we want to tag a whole resource, i.e., all triples that have the resource as subject or object instead of just a single triple, the same can be done to that resource's type. The process is exactly the same in all aspects, only with subclass relations instead of subproperty ones. This is a much simpler method than modifying each triple with the resource as subject, and is useful when we have reason to doubt the validity of the existence of an entire resource instead just a single triple.

4.3 Quality through Reasoning

Explicit metadata annotation of data quality is only one approach to the whole quality control problem. Often it's not feasible to keep track of the eligibility of data and instead pass the problem onwards. For example, in recommendation-based systems the system can keep track of the preferences of the user and the dates when those preferences are set. Then we can implicitly assume that older preferences are more likely to be of worse quality than newer ones, without having specific annotations about that information that would change day by day. This brings the whole data quality problem to a new level in this use case. Instead of having explicit knowledge about data quality we can reason that knowledge in real time from other information, such as the input date of the user preferences in this case.

The same situation arises in communal, distributed metadata creation. Optimally if any user can add information into a system, we would have some way to set a confidence value for that user. The system could keep track of that value and continuously update the system metadata accordingly, but maintaining this information can become cumbersome. Instead, we can assign this task to the reasoning logic as well: the confidence value of a user can be dynamically assessed based on info about that user. For example, in a culture portal such as CultureSampo [37] the confidence value of a user would be greatly boosted if we know that he is a professional in an appropriate museum or association.

Reasoning confidence values in addition to explicitly annotating them can be an elegant addition to a solution. However, it is not always possible—either we do not have enough secondary data on hand to accurately assess it, or the subject matter is such that reasoning confidence is not even theoretically feasible. For example, in black box type free natural text annotation against a reference ontology can be such a convoluted process that annotating the justifications for a confidence value instead of simply annotating the value is not practical. Transferring data from one system to another is also more complicated, since much of the semantic meaning of the

data is included in the *logic of the system* instead of exportable data, and cannot be transferred in a simple way. Other systems can come to completely different conclusions about the data with different logic rules. Such problems could also arise with explicit confidence values, but at least in theory they are easier to take into account, because theoretically all the information is present in the data.

5 Design and Implementation

This section covers the details behind the prototype system and work flow: the *how* of the solutions to the expressed problems, and also *why* these specific solutions to quality control problems described in the previous section were chosen. The focus is on technical low-level decisions and the justification of the choices made.

5.1 Quality Refinement Work Flow

To support the background covered in the previous sections, examples of real use cases and methods which can utilize data quality control are described. The aim is to show that real problems in the field of semantic web data creation can benefit from data quality control.

The aim of a system that utilizes data quality control is the efficient enhancement of data quality utilizing both automatic reasoning and user expertise. The starting data quality can be assessed using different machine reasoning heuristics, which give confidence value metadata as output. A domain expert can then assess the data quality by hand in priority order according to these heuristics, correcting errors as they appear.

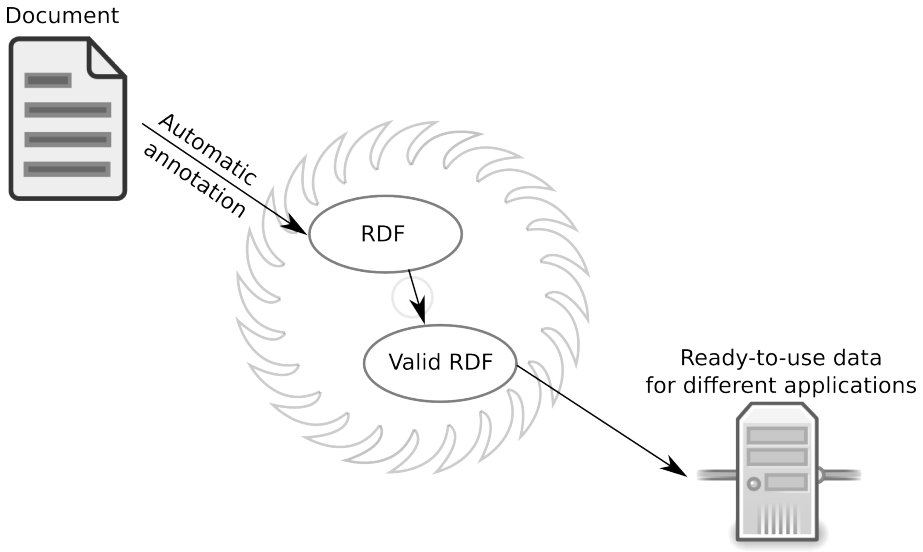

Figure 15: The route from documents to valid data

The basic process of creating new data for the Semantic Web is illustrated in Fig. 15. We begin with data in any traditional data format, such as plain text, XML, CSV or a relational database. Then, a direct transformation from the source format to RDF is applied. RDF is so expressive that, given proper parameters, practically any data format can have a relatively straightforward RDF transformation applied to it. However, because of the same expressiveness, there is a multitude of ways to express data as RDF, and a direct, mechanical transformation is often quite crude due to errors in the source data and the transformation process. The source code

for a direct example XML to RDF transformer with no data-specific assumptions is given in Appendix 7. It transforms any kind of XML data into RDF, changing the hierarchical tree structure of XML into triples for the graph structure of RDF.

To refine the newly created RDF, it is imported it into a framework that can flexibly be used to modify it in ways that improve its quality. The quality of an RDF transformation outlined above is one aspect to consider, but there are others as well. Any data format transformation is inherently quite prone to errors, and additionally the source data may contain erroneous information that manifests itself only after a data format change. The framework should be able to take all these things into consideration.

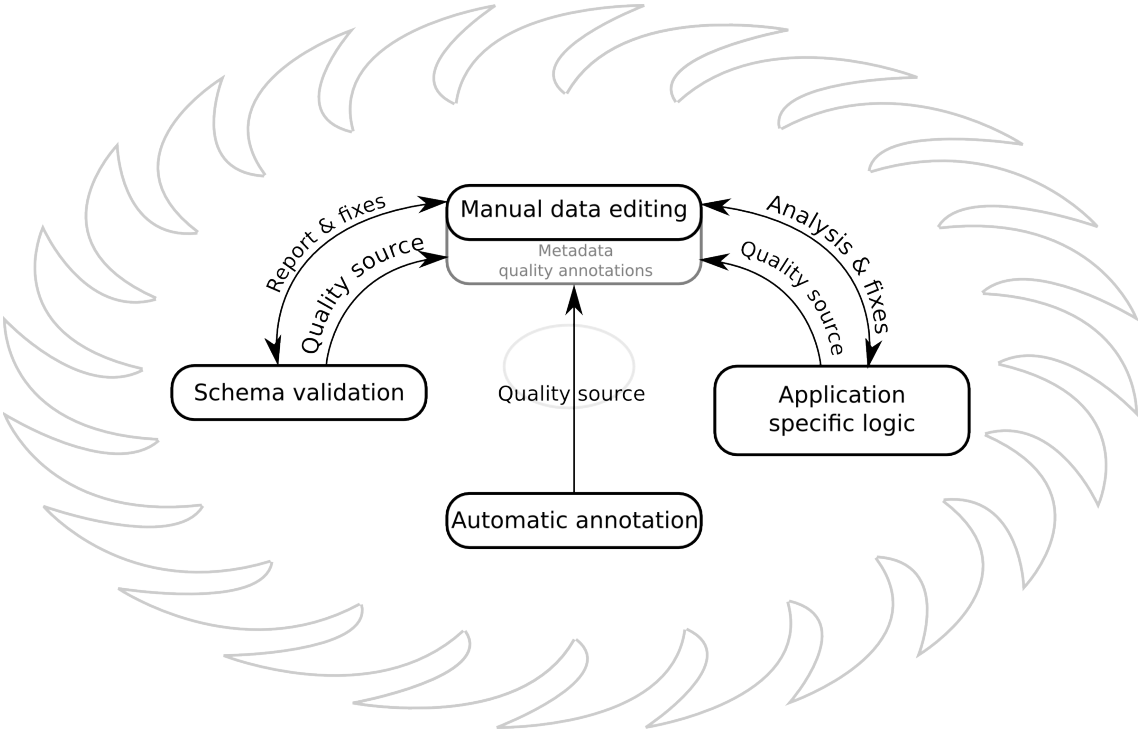

Figure 16: Structure of a quality control annotation environment

Different types of quality considerations require different modules in the framework to handle them. Fig. 16 outlines the basic inter-module interaction for improving data quality. The systematical crudeness of raw RDF must usually be handled with application specific logic, applied externally to the dataset through small programs or scripts. This is not part of the framework per se, but nevertheless the framework must be designed to allow this, by enabling easy exporting and importing of data. This logic can fix larger issues in the data, but it can also be a source of quality metadata, which in turn can be stored in the data graph with the methods described in Sec. 4.2.

If an automatic annotation system was used that provides quality information, that information can also be converted to quality annotations. This can be a major source of quality information, especially when creating new RDF data from other

data formats or directly from documents or plain text. As there are many different automatic annotation systems which give quality and confidence information in varying ways, each one requires a case-specific mapping to the used quality annotation system. Such a mapping isn't necessarily a complex one and should be domain independent in most cases, so developing one is not a big undertaking.

Often we have to work with RDF datasets that we do not know anything about in addition to the raw data. The origins, source format or more accurate semantics are things that are integral to developing application specific logic, so it's hard to generate quality information or fix the data directly based on our knowledge of it. While coming up with quality information based only on the data and the schema contained within is hard, it is possible to a degree. For this scenario when we don't know anything else, schema-based validation, as described in Sec. 4.1, is often useful. It can provide a solid starting point for refining the data quality and pinpoint potential systematic problems the dataset may have. Similar to application specific logic, the pinpointed items can either be directly fixed or simply stored as quality annotations.

In the heart of things is an editor with which users can edit any aspect of the data he wants, without restricting themselves to a single paradigm of improving quality, such as schema-based validation. Since the quality annotations are an integral part of the basic data, the data can be marked accordingly in the user interface as the user comes across a piece of data that is described with them. Manual editing is slow, but can plug all the holes the other methods can't access, and also directly utilize the quality annotations.

Finally, after the data is modified in the framework and deemed to be of sufficient quality, it can then be exported as a self-contained dataset and readily used in different applications. The specifics of how the system proposed here would work internally, and the services and interfaces required for the components to work together, are described in the next section.

5.2 Use Case

A good target for applying semantic web principles and technologies are information portals on the Internet that have much data, often user generated, that have related content but are not actually explicitly linked to each other through the use of hyperlinks and common categories. An example of such a website is Sosiaaliportti[15], a portal that provides both expert- and peer-provided information about child and social welfare.

Data from Sosiaaliportti provides for a good base for testing metadata quality control, because automated annotation has already been tested for it and compared to a baseline established by multiple human expert annotators.[49] The data consists of two types of documents: excerpts from the site's handbook for child welfare, and question-answer pairs from the consultancy service archive. The documents were annotated using concepts from the Finnish Ontology of Health and Welfare, TERO, by six domain experts and automated annotation (Sec. 5.5).

[15] http://www.sosiaaliportti.fi

The previous research on the Sosiaaliportti data was to compare how automated annotation compares to human indexing. The results were encouraging: automated annotation is quite good, but still not quite on the same level as domain expert humans can achieve. This is where quality control steps in: since we have a baseline quality for what the annotation should be, and an automated "best guess" annotation, we can provide an environment that enables a domain expert to easily smooth out the edges of the automated process, and ultimately prepare a good quality annotation with much less work than it would take for the human to do it from scratch. A good editor with a quality control aspect can make things a lot simpler and faster.

A somewhat similar use case with a differing paradigm is analyzing an RDF conversion of the Finnish Defense Force's dataset of norms and standards. [50] There, unlike with Sosiaaliportti data, the data is also initially structured. Nevertheless it can serve as an example of the problems that can arise when a supposedly well-structured dataset is subjected to a data format conversion. Instead of automatic annotation, manual fixing, application specific logic and schema validation are emphasized. This, alongside with directly creating new RDF data from unstructured text documents or other free-form formats, is the other main paradigm to which quality control needs to be applied.

5.3 Quality Annotation Markup

Handling quality annotations is a vital part of a system like the one described here, and the method of how it is done must be chosen carefully. Sec. 4.2 described various ways in which the quality of triples and resources can be annotated. Each have their specific strengths and weaknesses, and not all are suitable as a general solution, as is required here.

The chosen, novel method is to implement the functionality with hierarchic relations (Sec. 4.2.5): it offers an intuitive way for implementation in an editor environment with enough flexibility for various possible needs that might arise. It also isn't necessarily too complex on the graph level, so it doesn't clutter small datasets with a disproportionate amount of additional metadata.

The granularity, or the amount of accuracy levels of the quality annotation model is an important decision (Sec. 4.2). An intuitive, and often the first choice considered as a way to rate imperfection and confidence is with percentage values. This is also the approach taken by Preuveneers et al. in their OWL quality control extension. [4]

However, percentages are actually much too accurate if the whole spectrum is used—humans cannot make distinctions between different amounts of imperfection on such a fine-grained level. It has been found that just a handful of imperfection levels, whose relations to each other is known, is enough for human perception.[51, 52] Thus, if we need varying imperfection levels for triples with a certain property, it is enough to create a handful of uncertain subproperties for it. That handful can then be internally sorted in an order from the least uncertain to the most uncertain. The whole idea is depicted in Fig. 17. Different imperfection levels are arranged in relation with each other with subproperty of -relations (for triple-specific annotation)

and subclass of -relations (for resource-specific annotation), divided into different types of imperfection. Additionally, levels of imperfection for different types can be marked equal through `equallyValidTo`-relations, so even the different types of imperfection are somehow comparable.

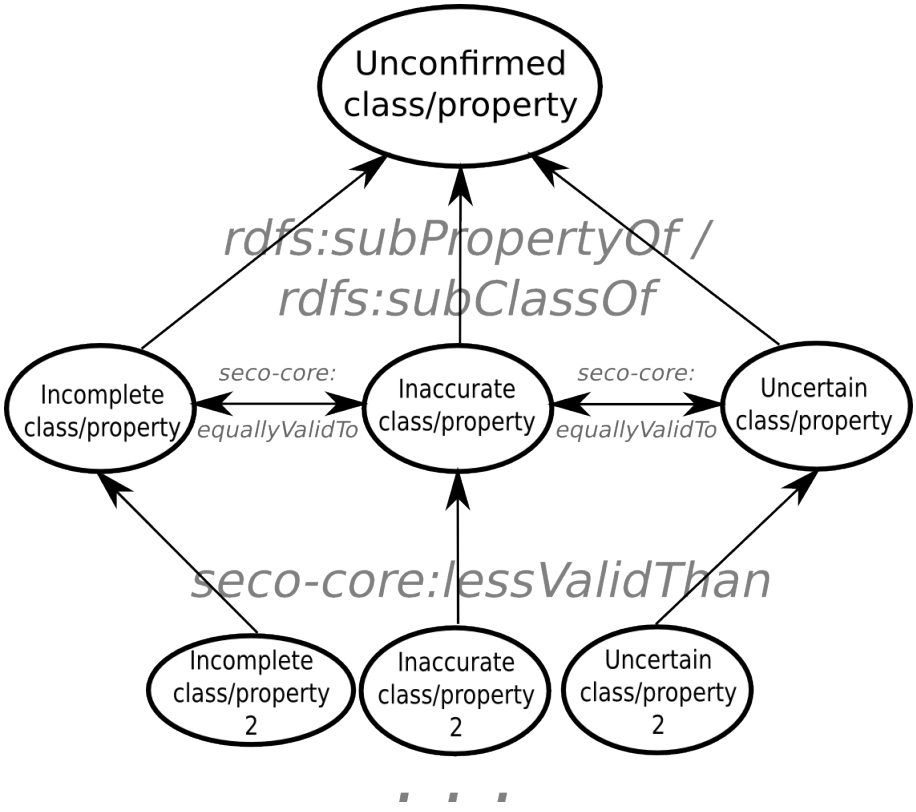

Figure 17: Different types of unconfirmed properties

With this kind of model, the complexity of the quality model is entirely customizable, regardless of the intrinsic properties of the data. This is made possible due to the fact that the model focuses on relative quality levels instead of absolute ones. This kind of approach loses some fidelity when observed out of context, but the level of imperfection some piece of data has is only really relevant when compared to some other data on another level of imperfection. Thus, the absolute imperfection levels are not as important as knowing the relation between the imperfection levels.

No explicit information is stored of what the original property was that got replaced. This means that when reversing the operation, all the superproperties are considered, so unconfirming a triple is not a trivially reversible operation without knowledge of what the property was before it. The reason for this behavior is that we can not be certain that the immediate superproperty is the originating one, and there can also be multiple immediate superproperties. Formally this means that the unconfirmation operation is a surjection and there is no straightforward inverse.

5.4 EMO: RDF Management Framework with Quality Control

Managing the input and output of our RDF data and the quality annotations in it is a large undertaking, and the answer proposed in this thesis is a multi-pronged prototype framework called EMO. Fig. 18 describes the architecture. Functionally, EMO is the data index behind the scenes, very efficient and scalable, plus the sum of the separate parts which create and modify the data within. In addition to being highly scalable, the data index also provides inferring, meaning that conclusions such as the equivalence of individuals and classes are calculated on the fly as data is being added.

In the core of the framework is SAHA [53], where the actual editing of the data takes place, and the quality annotations that go along with it. Interacting with it are three kinds of systems that act as a source of quality information for the quality annotations. Two of them, schema validation and and application specific logic systems, also directly interact with the data based on their view of the quality of the data.

EMO can also be thought about simply as a collection of synergic data-handling applications that share the same common data store. Below, the different applications that deal with the creation and modification of data (as per Fig. 18) are described in more detail.

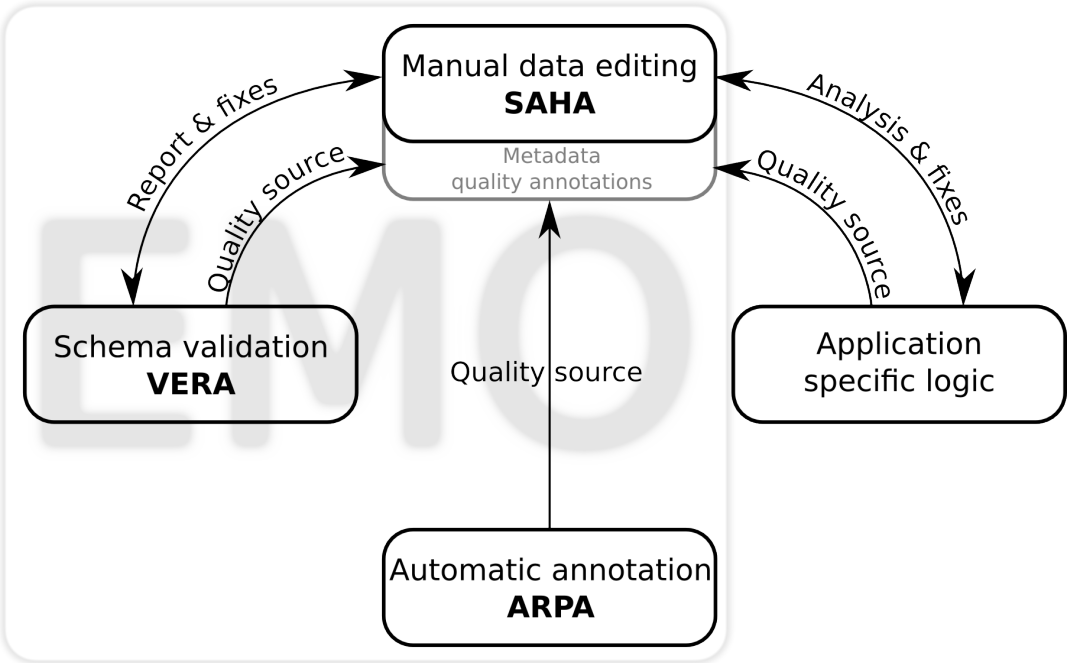

Figure 18: Metadata quality annotation management

5.5 ARPA: Centralized Automated Annotation

We often want to automatically match text documents against an ontology or vocabulary of some sort to get a basic machine understandable idea of what the documents are about and how their contents relate to each other. This process is called topic extraction, or automated annotation, and is a great way to bring semantics into applications based on traditional ones. [54, 55]

A centralized system for providing automatic annotations is helpful for efficient combining of many different autoannotation tasks to a single service. Such a system, called ARPA, is integrated into EMO, to be used either by any external source or another part of EMO, such as SAHA. ARPA receives parameters for annotation requests from clients, and then queries the back-end systems, engines, accordingly. Figure 19 shows the architecture of ARPA. The parameters of the REST API through which clients interact with ARPA are listed below.

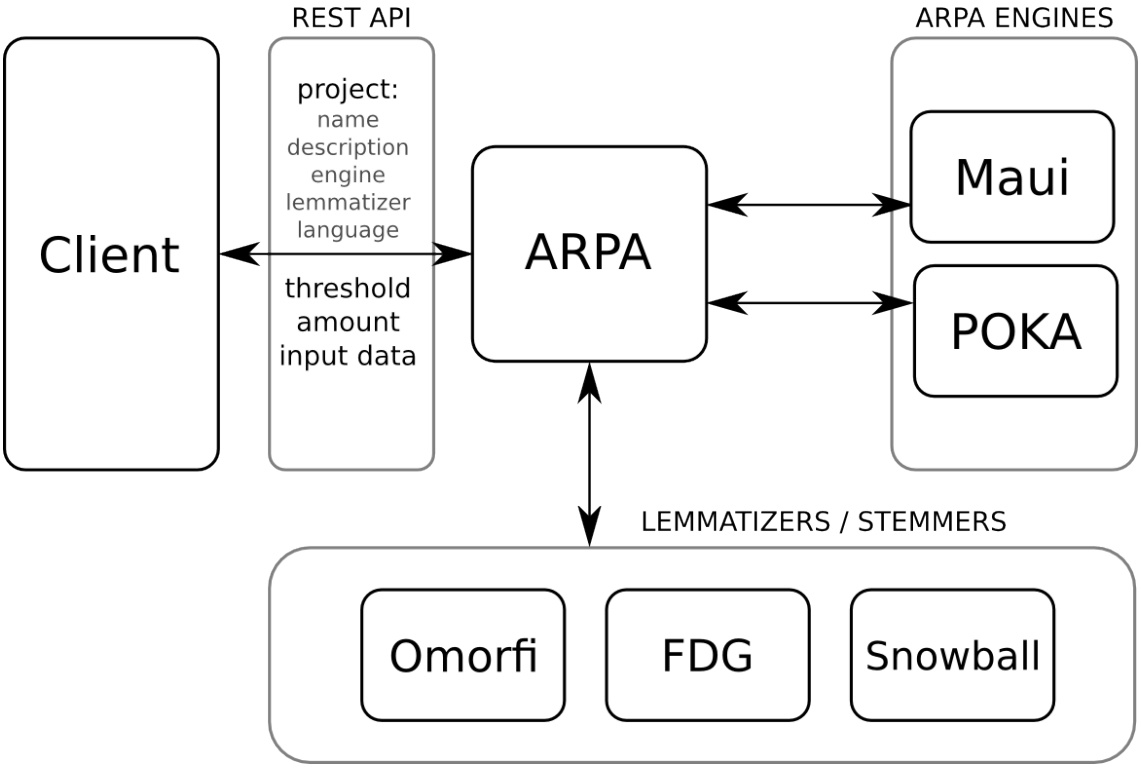

Figure 19: The architecture of ARPA

project

The main level of abstraction for ARPA is an ARPA project. It is intended to hold all linguistic details in one single entity that can be easily referred to in the API.

Name Name of the project. This is the main identifier of the project when referred to from the API.

Description A short description of the project.

Engine The part of the system that actually performs the annotation. Two main types of engines are used currently as can be seen from Fig. 19: Maui and POKA.

Lemmatizer/Stemmer The baseform services, i.e., the systems that handle the removal of inflection from the words of the input, are separated from the engine, even though some engines can handle baseforming as well. Currently, three different systems have been tested: Omorfi[16] (open source) and FDG[17] (presently known as Connexor's Machinese Syntax, commercial) for Finnish, and Snowball[18] for English and Swedish.

Language The language the project is meant to annotate. Generally, almost all engines and baseform services are tailored for a single language, and so this parameter is not actual input to the system. It is just meant to provide the language information to the user as part of the project description.

threshold The confidence threshold ($\in [0, 1]$) that serves as the minimum for the shown results. Note that some engines do not provide confidence information; in this case the confidence of every annotation is set to 1.

amount The maximum amount of results received. If the engine would otherwise return more than this amount, only the ones with highest confidences are shown. In case of tied confidences (such as when an engine does not provide confidence information) the choice is arbitrary.

input data The actual textual input to be annotated. There is no upper limit for the length of the input, save for the constraints of the HTTP GET protocol.

5.5.1 Autoannotation Engines

ARPA is just a front-end for annotating documents automatically. The actual concept matching is done by underlying subsystems, or engines. Many such engines exist, and different ones focus on different things. Some focus on simplicity and performance, while others aim for complexity in order to reach better annotation quality. Two systems with relatively orthogonal approaches are currently coveredin ARPA: Maui topic indexer [55] and POKA information extractor [56].

Maui is a indexing system whose main purpose in the context of automatic annotation is to extract the most relevant ontological concepts from a text document using a pre-built model based on authoritative learning data [55]. The teaching process requires several (preferably around 50) example documents and good-quality annotations for them. Using the example annotations and the ontological structure of the provided vocabulary, the model to automatically annotate future documents

[16] http://home.gna.org/omorfi/omorfi/
[17] http://www.connexor.eu/technology/machinese/
[18] http://snowball.tartarus.org/

is built. This model is then used to give an individual confidence value, among other metrics, to each ontological concept found in a document. The internal logic of Maui is built on previous keyphrase extraction systems, KEA [57] and KEA++ [58], and the implementation is based on the WEKA machine learning toolset [59].

POKA is a general information extraction system, with focus on functional simplicity and extensiveness over quality. Unlike Maui, it does not require teaching, which makes adopting it easier. On the other hand, this means that the results are not graded by confidence, making it hard to choose just a handful of the most relevant concepts pertaining to a document. In addition to ontological matching, POKA also does named entity recognition (NER), finding entities such as people and places in the processed documents. [56]

5.5.2 Maui Confidence Markup

It was stated earlier that a very fine level of detail for confidence ($\in$ `[0,1]`) values is unnecessary for human perception and only clutters the metadata model (Sec. 5.3). However, some sort of hierarchical confidence prioritization must still be made. For annotations that use Maui as the ARPA-engine, such a hierarchy was made based on the performance of the Sosiaaliportti Maui project (Sec. 5.1) against a set of test data, shown in Fig. 20. The figure shows the amount of correct and incorrect (i.e., differing from the human annotators) Maui annotations. Noting the logarithmic scale, we can see that there are a lot more results with low confidence than a high one, and that the precision increases steadily as the confidence rises, as expected.

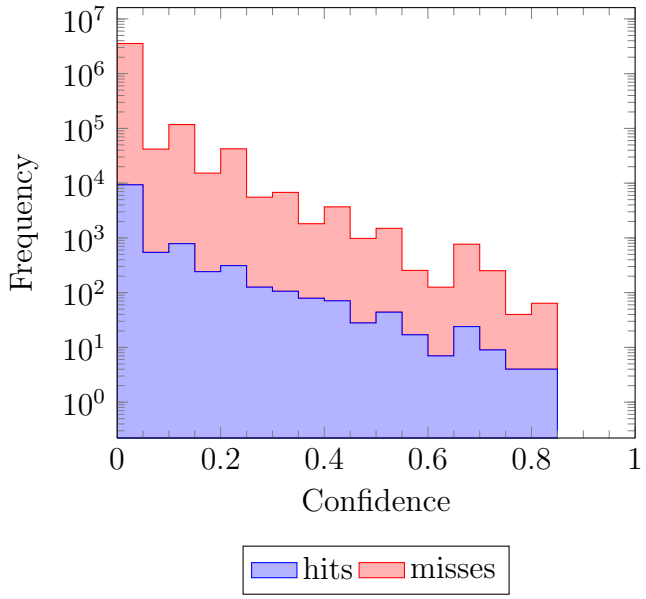

Figure 20: Frequency of hit/miss Maui annotations

Fig. 21 shows the percentual precision of annotations by confidence. The best precision overall acquired in the testing process was around 40%, which is already quite close to the precision between different human annotators [49], so all anno-

tations above that can be classified together as the least improbable class. Also, it is useful to have the classes with lowest confidence as the most common, so the better quality guesses can be used as a higher level abstraction level. Moreover, it is expected that for every good guess there is a multitude of bad ones.

Generally, assuming an exponentially diminishing class population (as can be seen from Fig. 20) and a continuous distribution, the confidence thresholds $T_x, x \in \{1, 2, \ldots, m\}$ are formally defined by

$$\begin{cases} P(X \leq T_x) \leq \frac{\sum_{k=1}^{x}(m+1-k)^p T_h}{\sum_{k=1}^{m}(m+1-k)^p} \\ P(X \geq T_x) \geq 1 - \frac{\sum_{k=1}^{x}(m+1-k)^p T_h}{\sum_{k=1}^{m}(m+1-k)^p} \end{cases}, \tag{1}$$

where p is the parameter that controls the relative class sizes (which nevertheless are exponential) and T_h is the threshold when the precision is comparable to human annotators (note that $T_m = T_h$). The equations originate from the definition of *median* and *percentiles* for a continuous distribution, but instead of equally spaced percentiles, the equations give exponentially shrinking ones for growing values of x, so the lower levels are more common than the higher ones.

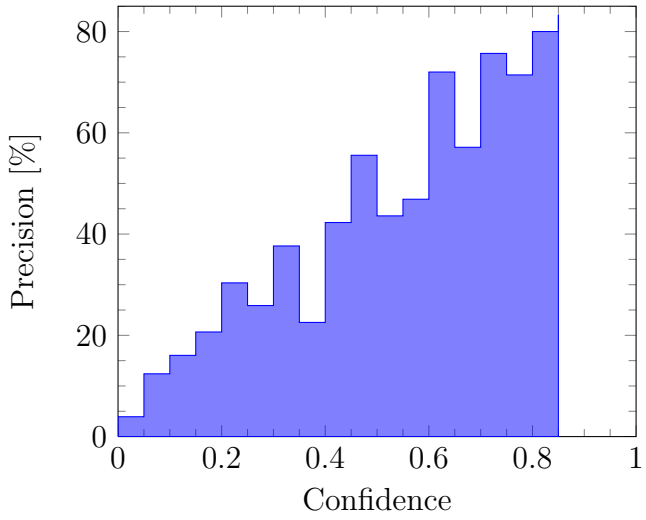

Figure 21: Precision of Maui annotations by confidence

Computed for our sample data with $m = 3$ (to yield 4 hierarchy levels separated by 3 thresholds) and $T_h = 0.4$, this gives us the thresholds shown in Table 1.

Table 1: Maui confidence thresholds

x	threshold
1	0.029
2	0.200
3	0.402

This is only an exemplary mapping from an arbitrary data quality representation to the proposed quality annotation system in RDF. It is presented as proof of concept of the usability and versatility of the metadata model in the context of data conversion. This is an important aspect to consider since the creation of new data is integral to the development of the Semantic Web. However the mapping is not only a throwaway example as it solves a common use case in this area: converting percentage-based quality information of an established automatic annotation system into the proposed quality annotation metadata model (Sec. 5.3) with a semantically sound justification.

In the end we would have RDF data that has its quality annotated triple by triple with the markup described in Sec. 5.3, divided to four different levels whose appropriate properties are separated by the `lessValidThan`-relations depicted in Fig. 17. Then we can proceed to assess the data in priority order, starting from the level with least confidence, as given by our automatic annotation process.

5.6 SAHA: Collaborative Metadata Editing

SAHA is a WWW-based tool for distributed creation of instance data for the semantic web in RDF form. It is designed from the ground up to be robustly usable for collaboration, without people interfering with each others work. This is very important in any collaborative tool which aggregates clusters of triples as some kind of entities (such as resources) to be worked on.

The basis features of SAHA include:

- Creating, deleting and editing of resources denoted by URIs in a robust distributed web environment
- Searching instance either by their ontological class or with a global autocompletion search
- Inserting ontology concepts for defined fields from external ONKI ontology services [60]
- Creating and editing point/area/route location data with a map interface
- Accessing the data from a SPARQL endpoint [61] or other REST and web service APIs
- Exporting the data as RDF/XML, Turtle (Notation 3) [13] or N-Triples
- Searching data with an integrated single object property based customizable facet-search engine, HAKO [53]

5.6.1 SAHA Technical Architecture

The general system architecture is shown in Fig. 22. The central component in the architecture is the MVC framework. As the MVC system, the Spring Framework[19]

[19] http://www.springsource.org/

is used, and Freemarker[20] is the template engine. Both these and the business layer code are written in Java. It receives page load requests, queries the persistence store, acquires the correct template and fills it with the right data. In addition to normal requests, the MVC system handles asynchronous requests used throughout the system in the form of DWR calls.[21]

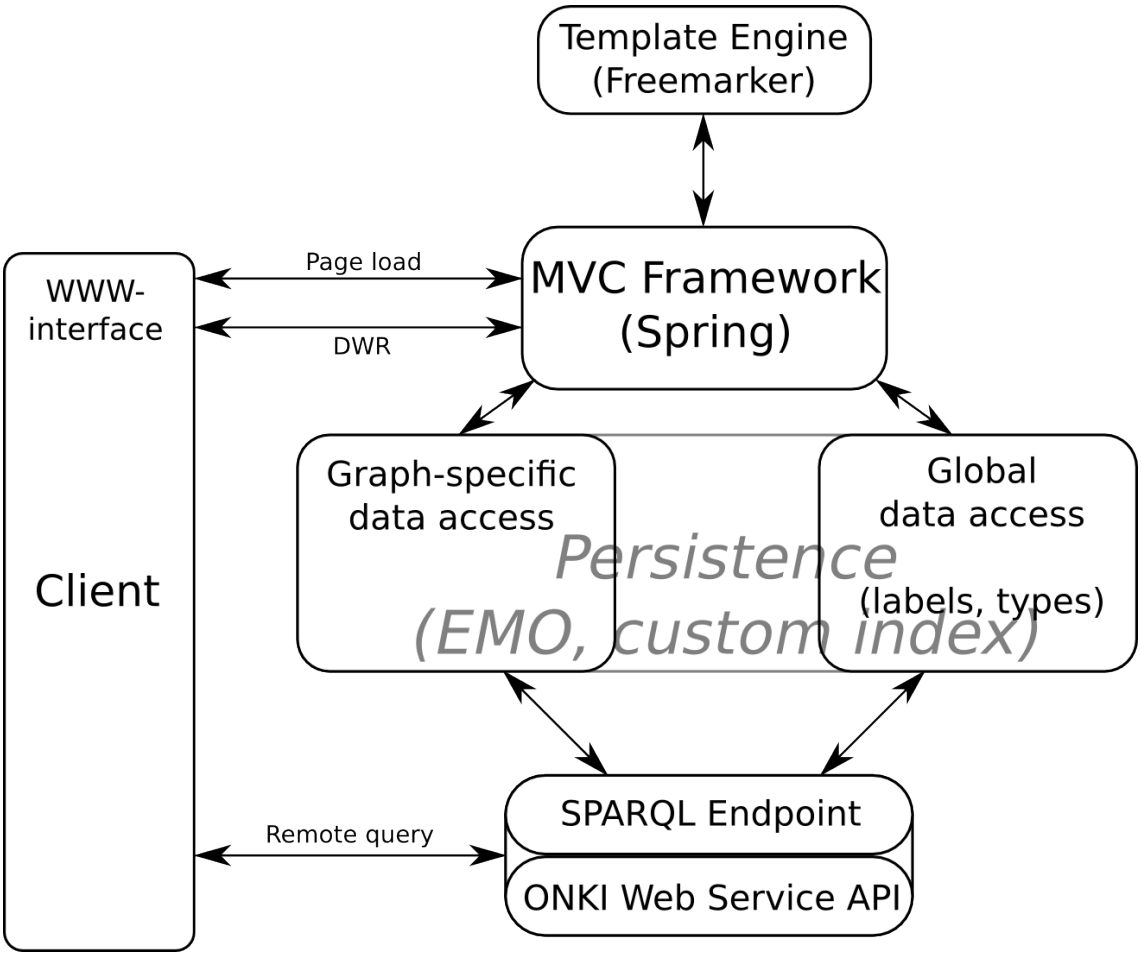

Figure 22: SAHA system architecture

Data access from the website is twofold. All the data is stored as triples in the custom triple store, which maintains both a general store and separate indices for common operations such as label and type searches. It is also monitoring changes and inferring in real time, meaning that all triples added or modified are checked for possible implications concerning reasoning—some types of triples that require additional actions upon inference are resources marked as identical through `owl:sameAs`, and triples that are part of an auxiliary index, such as resource labels. Exactly what is considered a label, in addition to commonly known label properties such as `rdfs:label` can depend on the data and is configured separately.

[20] http://freemarker.sourceforge.net/
[21] http://directwebremoting.org/

An alternate way to access the data is through a SPARQL endpoint. Custom global or graph (project) specific queries can be executed to the triple store directly. Another way to retrieve data from a graph in the triple store is to use the ONKI [60] web service API. This is quite useful as then, in addition to regular ONKI ontologies, projects in SAHA can be queried from other projects through the regular ontology autocompletion search for object properties. Similarly, any site or application that makes use of ONKI ontologies through its web service API can utilize SAHA projects in a similar manner.

5.6.2 SAHA Quality Control Tools

To actually modify the quality annotations described in Sec. 4.2, the user interface of SAHA must have specific views for this purpose. Optimally, these views would be tailor-made to suit the most common tasks the user faces when manually editing the quality information. Two such ways to interact with the annotations are implemented in SAHA.

The main view concerning the quality annotations is the general unconfirmed data view, shown in Fig. 23. It shows all the triples annotated as unconfirmed in the data, grouped by subject; often the same resource has many unconfirmed pieces of information about it. The resources are listed in priority order as per the model described in Fig. 17, with resources containing the least probable triples listed first (alternatively, the amount of unconfirmed triples a single resource has could also be a factor). This allows the user to check the validity of the "biggest offenders" first, as those are the ones that most probably need fixing.

The *fixing* of a triple works as follows. As explained in Sec. 4.2, the validity of a triple is controlled through the subproperty relations of its predicate. All the superproperties for the predicate that are also not subproperties of the general unconfirmed property are computed, and offered as choices for the user, like *name* and *label* in Fig. 23. When a user chooses one of these choices, the property of the triple is replaced with that choice, and thus the unconfirmed state is ended for that triple.

The other view concerning quality annotations is showing them alongside with other data when browsing data normally. This behavior can be seen in Fig. 24. Unconfirmed triples are marked with a distinct color, and the same fixing functionality is offered here as in the general view. This allows the usage of quality annotations alongside regular data management operations: if the user happens to come across an unconfirmed piece of data when creating or modifying normal data, he may assess it at the same time.

5.7 VERA: Schema Validation and Data Integrity Checking

VERA is a system that generates quality and validation info about RDF data when compared to its schema defined in RDFS and OWL. The generated info is compiled as a report presented in HTML. In this work VERA is used as an integrated module in the SAHA metadata management environment, but it was originally designed and

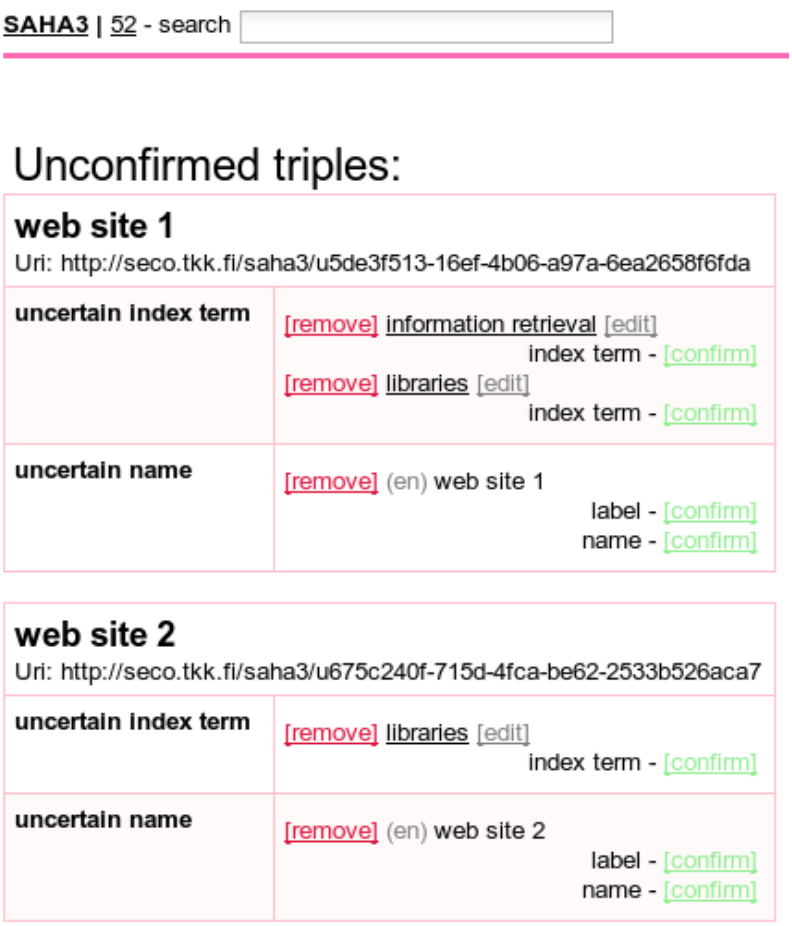

Figure 23: General unconfirmed data view of SAHA

implemented as a standalone application that functions both from the command line and through a web interface.

5.7.1 Report Structure

The report is essentially a list of items that each represent a potential issue with the data, divided into different categories by severity: errors, warnings, notifications and information. The report is further divided into items about validity and integrity. Validity items mostly have to do with the relation between the schema and the data. They show discrepancies about the schema-based assumptions of the data and the actual data, for example the objects for triples based on the restrictions given in the schema. Integrity items concern the general structure of the RDF graph, notifying about things such as orphaned resources: resources with no incoming or outgoing links to other resources.

It is important to note that none of the items listed by VERA need not necessarily be actual errors. Rather, it gives a list of possible problems that an expert user can assess and modify the schema or data if needed. Many of the items are not actual

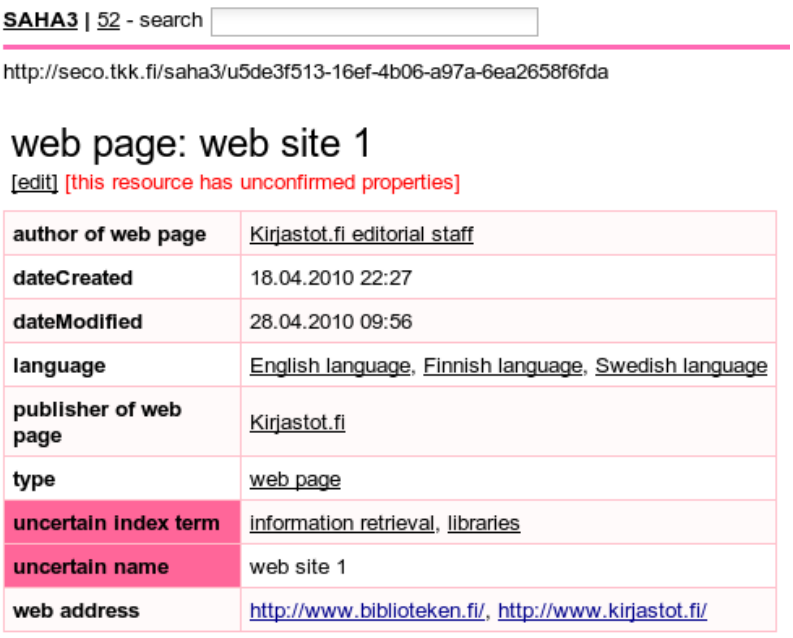

Figure 24: Unconfirmed data in the resource view of SAHA

problems even in the majority of use cases. Below, different types of items are listed, along with a short explanation detailing when and why the item appears in a report. An example view of a VERA report is shown in Fig. 25, for a piece of an older version of the BookSampo [62] dataset, validated here for testing purposes. Figures 26 and 27 also depict real life examples from the same dataset.

Errors

Errors are the worst class of report items, and can be directly considered marks of erroneous data regardless of semantic interpretation.

Illegal object node type on a triple

The most common form of a serious semantic (as opposed to syntactic) error in RDF data is having literal values on properties that are marked as object properties or vice versa (mainly by being of types `owl:ObjectProperty`[22] or `owl:DatatypeProperty`[23]). Many applications assume that all values on object properties are indeed resources instead of literals, and will cause errors if this is not the case.

A subset of this type of item is further highlighted: if a `type`[24] of a resource is set as a literal instead of a resource value, it provides a separate report item.

[22] http://www.w3.org/2002/07/owl#ObjectProperty
[23] http://www.w3.org/2002/07/owl#DatatypeProperty
[24] http://www.w3.org/1999/02/22-rdf-syntax-ns#type

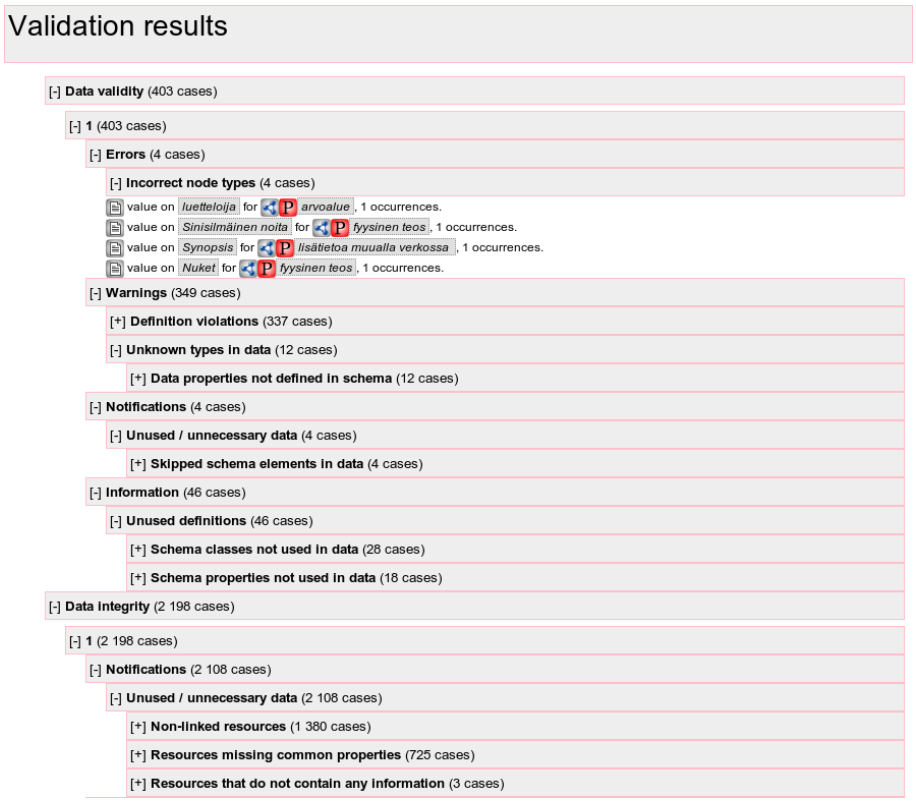

Figure 25: An example VERA report

Malformed definitions

The interal consistency of `owl:Restrictions` is checked, and a report item provided if something unexpected is found. Examples of such occurrences are a cardinality value on a restriction that is not an integer, a restriction with no `owl:onProperty` definition, or a literal found when a resource was expected, for example as a part of an RDF list.

A restriction in the schema is not used on any class

Even though it is not an error in a strict sense, an error report item is provided if a restriction does not pertain to any class: formally, no class is a subclass of the restriction.

Warnings

Warnings are mostly notes about how the data conforms to the schema, and what the validation system thinks are the parts in the data that conflict this conception.

A class/property in data that does not exist in schema

There are situations where you thoroughly want to control which classes and properties are used in the data you have. Hence, a warning type item is generated if the data uses a property and class that is not explicitly

defined. In most applications at least the type should be defined explicitly for each resource used as a class or property.

Domain/Range violation

Special attention is given to the `rdfs:domain` and `rdfs:range` definitions in the schema, whose semantics were described in Sec. 2.2. A domain violation item is generated for each occurrence where a property is used on a resource whose type is not defined as its domain in the schema. Likewise, a range violation is generated for each occurrence where a property value does not match the expected pattern from its range definition in the schema. This interpretation is not what those definitions semantically mean (as explained in Sec. 2.2), but is useful in the context of validation and used in other applications in the same area as well [63].

The warning report generated is more complex than most others. Instead of a single row or information, three distinct aspects of the items are listed. The first is a list of the instances affected by the property/type combination the item is about. In the integrated SAHA environment this provides for easy access to go and fix the problems where they exist, should that be necessary. The second is a list of the properties that would make the triple valid if the current one was substituted with one of them.

The last is a list of the types for the triple's subject (for domain violations) or object (for range violations) that would make the triple valid if the current one was substituted for them. The last one is not unambiguous, because there might be multiple required types through multiple domain/range definitions, possibly through inheritance. Also there might be optional type requirements if the domain/range definitions are unions instead of simple resources. The arbitrary type pattern is computed and shown in the report. An example domain violation report item is shown in Fig. 26, which shows that 7 out of the 45 occurrences of the property *kuvittaja* (illustrator) have a subject of type *kansi* (cover), which goes against the schema definition of what the type should be. In this case the schema says that the instances should also be of type *physical work*, hinting that there could be an issue with the class hierarchy, since *cover* was not also found to be a *physical work*.

Literal value doesn't conform to range (XSD datatype or regexp)

Technically a subset of range violations, ranges on literal valued triples are handled slightly differently. If the range consists of *XSD datatypes* [64], the literal is validated to make sure it conforms to those datatypes. There is also an experimental way to have regular expressions as range definitions, but this functionality is currently not used and thus a description of it is not included in this work.

Restriction violation

All of the `owl:Restriction`s in the schema are interpreted strictly and if the data would require additional triples to conform to them, an item is generated. The restrictions checked are owl:maxCardinality, owl:minCardinality,

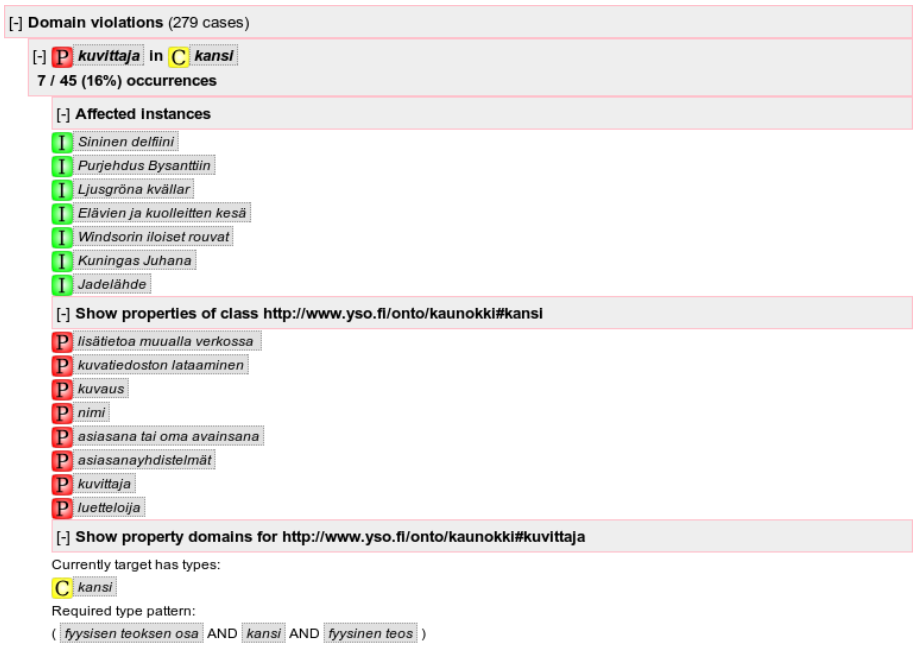

Figure 26: An example domain violation in a VERA report

owl:cardinality, owl:allValuesFrom, owl:someValuesFrom and owl:hasValue. The item shows the offending value and the range of correct values.

Notifications

Notifications are items that can sometimes be interesting, but are usually not errors by themselves per se.

Resource has no information besides type

A resource with no `rdf:type` definition generates a notification. According to OWL semantics, every resource should preferably have a type defined somewhere on the data graph. [6]

Orphaned resources

Resources in data with no references in or out to other resources generate a notification. As the aim of RDF data is to provide an interlinked web of data, orphaned resources are often unwanted and useless, perhaps remnants of deprecated data constructs.

Data redefines schema elements

If the validation is run in a mode where the schema and data are separate models instead of a single one, an item is generated for each entry where a schema item is also defined in the data. It is not checked whether the definitions differ, only that both the schema and the data define the same resource.

Schema redefines core RDF resources (RDF, RDFS, OWL)

Another item that can at times be considered unwanted, is redefinitions

of the core RDF resources in the schema. Formally, the item is generated if the validated RDF contains triples that have a core RDF resource, such as `owl:Class` or `rdfs:domain`, as subject. If such triples have ended up in the data by accident, it can have vast and unpredictable consequences on machine reasoning.

Resource used as object but not subject

Sometimes we want to explicitly give information about each resource referred in the data. A notification item is generated for each resource that is used as an object in a triple, but does not have any explicit definitions, such as a label or a type.

Information

The Information section contains items that are not considered faults in the data by the validation process, but that might nevertheless be of interest to the user.

Unused schema definition

This item is generated if there are classes or properties defined in the schema that are never used in the data. This information could be useful for finding deprecated or otherwise useless schema definitions that can subsequently be deleted.

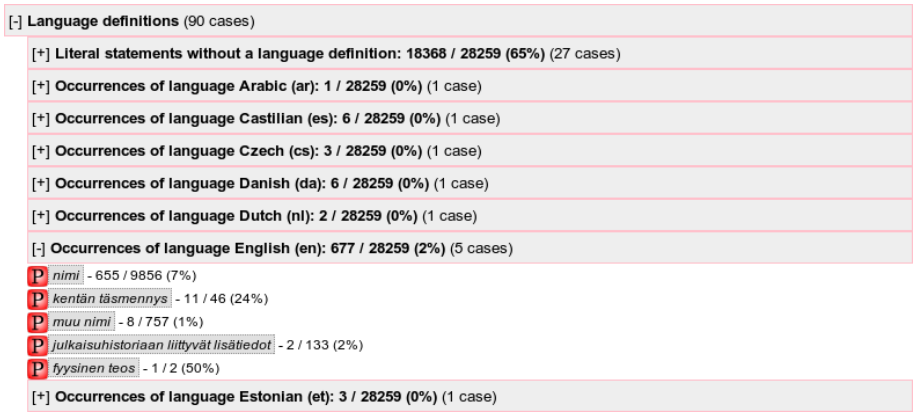

Figure 27: Language definition list in a VERA report

Language definitions

The report details the use of different language definitions for the literals in the data. Based on the `xml:lang` definitions in the data, the different languages used are listed, along with a list of how big a proportion of each property uses each language: Fig. 27 illustrates the matter: each language present has its own entry, including literal triples with no language definition. It tells how big a proportion of the literals are in that language overall, and the same separately for each literal property. In the figure we can see that, e.g., 2% of the literals in the data overall and 7% of the *nimi* literal values are in English.

5.7.2 Example Report Items

The report items given by VERA do not represent clear data errors except in some extreme cases. It is always up to the user and domain expert to assess them to ascertain that there really is a problem that needs fixing. To demonstrate this process, here are some example report items given by VERA when developing and testing the Finnish Defense Force public norm dataset [50] and smoothing out the rough edges of its RDF model. At this stage, the dataset was directly transformed from XML to RDF, with minimal dataset-specific considerations. Systematic errors in such transformations are very common due to anomalies in the source data and the actual transformation process logic.

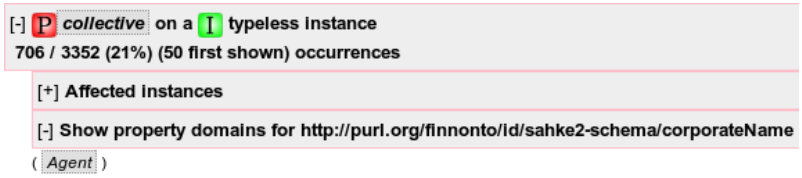

Figure 28: Example domain report item

Fig. 28 shows a domain violation report item in a VERA report. It is triggered by the fact that typeless resources use properties (*collective*) that have a domain definition (*Agent*) listed in the schema. Namely, a certain type was expected for the resource due to the domain definition, but none was found in the data. This is a good example of an issue that either could be a problem, or it could be the data working completely as intended. In this case, closer inspection reveals that all the 706 typeless resources in question are indeed Agents as suggested by the domain of the property used in them, but without explicit typing. Thus, a simple reasoner could have generated the type triples for us without generating erroneous data. However this is not always the case and explicit typing is often a fair requirement—hence the validation report item.

Figure 29: Example integrity report item

Fig. 29 shows a general report item about common properties (labels and types)

missing. Part of this item stems from the same root problem as the previous one—it is relatively common for one root cause to manifest itself in many different report items. From this item we see that in addition to the resources concerning Fig. 28, there are other resources without a type as well. The list below shows that while most are without a label, some are not, so the group of typeless resources is diverse. Should the user want to fix some resources by hand by explicitly typing them, it can be done easily by navigating to the editor view directly from here by clicking on their respective URIs. Since there are so many, the more feasible way is to use *application specific logic* as per Fig. 18 and fix the situation with a small program or script.

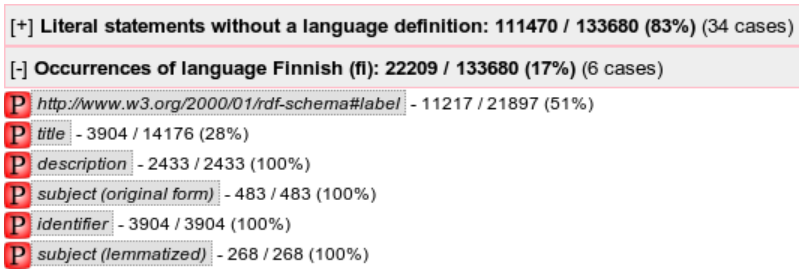

Figure 30: Example language report item

From a data integrity perspective, even semantically sound data with a healthy schema-data relationship can have much room for improvement. One example of this is language definition usage for triples with literal values. Optimally, every literal that's specifically written in some language (i.e., not a numerical value or an accepted common name) should have a language definition. Usually, values of a certain property are either all written in a specific language or none of them are. In Fig. 30 we see the language definition analysis of the data: only 17% of the literal triples have language definitions, and there are two common ones (title and label) with some but not all marked with a language definition. This could warrant a reconsideration of the language definition usage of the dataset. Sporadic language markings is usually not problematic in a single dataset, but when combining data from multiple sources annotated in multiple languages, the importance of them rises considerably.

6 Conclusions

Controlling data quality on the Semantic Web is a colossal undertaking, and the problem cannot be solved with a single work alone. This work proposes solutions to some aspects of the problem, both on a theoretical and practical level. One focus is the very method of annotating quality-related metadata in RDF, to which multiple solutions are offered with one being chosen for the implementation and refined further. The other focus is a framework that can utilize this kind of annotations. The basis for this framework is provided, along with a prototype implementation and qualitative analysis to justify the choices made.

To the research questions posed in the introduction, the following conclusions can be made:

How can data imperfection be taken into account on the Semantic Web?
Knowing about data imperfection has intrinsic value, even if we do not explicitly use that knowledge right away. The Semantic Web is about storing and combining pieces of information, even when they are not immediately useful for the task at hand. In the open world, any sensible piece of data can be useful to some other agent.

Naturally, if we can efficiently find out the imperfections of a dataset and fix them, this raises the value of our data. Finding underlying data imperfections is generally hard for RDF datasets, but machine reasoning can alleviate the problem to an extent. The methods differ from those used for more traditional data storage, but some exist for many stages of the data life cycle.

There are also many direct ways in which known data imperfection can be taken into account. Known unreliable data can be weighed as less relevant in reasoning or recommendation systems. It can also be found easily for modification or deletion, or omitted temporarily for certain tasks. In addition, all the different types of imperfect data listed in Sec. 2.8 can be handled differently as needed. Contextual quality is a special case: it can viably have application specific tailor-made logic defined to handle it specifically, while not affecting functionality oblivious to that kind of quality annotation at all.

How can data imperfection be annotated as metadata for RDF-based datasets?
This question covers the crux of the theoretical part of the work, described in full detail in Sec. 4.2. Many ways to annotate quality metadata in RDF are presented, with many different paradigms to the same problem. Some are more universal than others, while some are tailored to fit a narrower range of problems.

One novel approach, annotation with hierarchic relations of elements, is picked out as a suitable approach for the purposes described in this work. It uses ad hoc subproperties (for triple-specific annotation) or subtypes (for resource-specific annotation) to annotate metadata directly to the same RDF graph. The approach is implemented as part of the practical section of the work, to show prototype functionality that can be implemented with quality metadata.

The method was chosen because of its versatility and semantic validity. It's versatile in the sense that it can be used for most imaginable use cases for quality control while not forcing the user of the data to use it or know anything about it. Meanwhile, utilizing multiple inheritance is an elegant solution to the kind of data markup problems where a piece of data has characteristics of widely varying concepts.

How can data quality control be utilized in automatic or manual annotation?

There is no single all-encompassing answer to this question. The aim of the majority of the practical section in this work is to offer a single, possible answer, as the developed framework utilizes quality control in multiple places, both in automated and manual tasks.

For automatic annotation, quality control can be used to give quality information in addition to the achieved results, should the annotator provide them. Then the quality metadata can be used for any purpose, some of which are listed under the conclusions for the first research question above. They are of special use in semiautomatic annotation where a human ensures the quality of the automatic annotations, as then the ensuring process can be prioritized to handle the most probable errors first. A prototype combination for producing quality-annotated RDF from the Maui automatic annotator is presented, along with a way to produce useful quality rankings based on the linear confidence values an annotator system gives as output. The ranking is based on a single use case, but can also be used globally. This is due to the mathematics behind it being quite general, and the system behind the prototype data being tested as quite domain independent. [49]

For manual annotation, most of the same benefits and uses apply similarly. Furthermore, Web 2.0 applications that utilize crowd sourcing and distributed collaboration can benefit directly from explicit quality control. The reliability of each collaborator can be weighed and the data they create annotated accordingly as trusted or untrusted, an example of *contextual quality*.

The kind of quality control functionality described here could be applied to many kinds of semiautomatic or manual annotation processes, but not specifically as a complete solution for RDF datasets. Having the quality metadata in the same RDF graph as the data allows for greater portability and simplicity of the tools used. The data can be reused anywhere, with the quality annotations seamlessly integrated as a part of the dataset.

In addition to explicit quality annotations, schema validation (Sec. 4.1) similar to XML Schemas or relational databases also has a place on the Semantic Web. Special precautions and presumptions must be taken since the Open World Assumption (Sec. 2.5) is often in effect for the datasets in question. Schema validation is useful if a dataset is self contained on some level. Even then a validation report is not a list of errors per se, but rather a list of potential issues a domain expert can assess and fix the data accordingly if needed. There is a strong similarity to the confidence values an automatic annotator can present:

they are only the system's guess based on some heuristics and it is up to a human to scrutinize the potential problems.

There is yet much research to be done in the area of quality control on the Semantic Web. Analyzing the ideas and implementations of the thesis and any possible problems they might have is prudent. For known possible issues, some solutions are presented below which could provide a foundation for future research on the subject.

A shortcoming of the quality annotation model developed in this thesis (described in Sec. 5.3 and more thoroughly when implemented in Sec. 4.2) is that the operation of marking a triple as imperfect doesn't have a unique inverse, i.e., the mapping is not a bijection. Like in Fig. 23, since the unconfirmed property has multiple superproperties (*name* and *label*), we cannot know which one was originally used for the triple before it was annotated as imperfect. Generally speaking, the consequences of annotating a triple are bigger when the operation can not be easily and instantly reversed. This matter could be addressed in a couple of ways. The deannotation process logic could be remade to only accept a single superproperty. This does weaken the versatility of the annotation method however, as then there is no intrinsic support for the common case where a triple is fixed by swapping its property with a more generic superproperty. Alternatively, the framework could be expanded to provide an explicit bijective function for annotating each property, possibly with a separate data model to keep track of these relations. This would fix the issue, but would make the method much more complex.

As for the implementation of the quality control tools, multiple small improvements could be made. The theoretical introduction of the quality annotation method in Sec. 5.3 notes that the same method can also be used on a resource level and not just triple level. Currently only the triple level functionality is implemented as part of SAHA, but implementing the resource level control would provide additional benefits: the use cases tested so far show that resource-level imperfection annotations would indeed be useful. Also, the user interface could be implemented to indicate different validity levels with different styles, such as different colors in the resource view (Fig. 24).

The range and extent of RDFS and OWL constructs covered by VERA in its validation results are not clearly defined. For example, some OWL Full constructs are covered, while some OWL DL expressions remain uncovered. Examples of such expressions that are currently not covered by VERA are `owl:disjointWith`, `owl:FunctionalProperty` and `owl:InverseFunctionalProperty`. Checking for the expected usage of most of these might be of interest in some cases. This would optimally be rectified by making different validation processes for different OWL levels: validating a model as, e.g., OWL DL would be relatively straightforward and useful. The most extensive validation mode should optimally include all possible OWL expressions that might interest the user.

The framework as a whole, unlike the regular version of SAHA, has not seen much production use as of yet. Mainly this is due to much lower stability than the simple architecture of the standalone version. In order to vastly improve efficiency over

regular data stores such as the one used in standalone SAHA, the index system used for data storage in this framework (Sec. 5.4) is much more complex, and originally designed mainly with read-only usage in mind. This approach is less robust and less tested, resulting in more data inconsistencies and errors. The testing and fixing these minor stability bugs is an ongoing process but at present there is still much to be done in this area.

Perhaps the area where this work is most lacking is robust and quantitative user experience testing. In particular, it would be interesting to see how common annotator tasks are done from start to finish with and without quality control, and how it affects the methods used and the efficiency of the work. However, designing a suitable test scenario and performing the testing would take a lot of effort and be very time consuming, and as such could not be incorporated into this work. What was done instead was thorough reasoning behind the choices and smaller scale test use without a methodological testing procedure.

7 Acknowledgements

The described implementation relies on and is complemented by many subsystems developed by other members of the Semantic Computing Research Group in Aalto University[25].

- The data index behind the framework (see Sec. 5.4) has been developed by Eetu Mäkelä and notably used as the data store in the CultureSampo cultural heritage portal [65].
- The ARPA engines are work of their respective authors; see Sec. 5.5 for details.
- The first version of the current implementation of SAHA (see Sec. 5.6) is originally developed by Jussi Kurki [53]. Some of the design owns homage to the previous SAHA versions, implemented by Onni Valkeapää [66], Olli Alm [67] and Katariina Nyberg.

Parts not mentioned above: the VERA validation system, the ARPA interface, much of the functionality of SAHA including all quality control aspects, and the quality annotation model are original work by the author.

[25] http://www.seco.tkk.fi/

References

[1] R. Cyganiak and A. Jentzsch. Linking Open Data cloud diagram. `http://lod-cloud.net/`, 2011.

[2] Z. Ding, Y. Peng, and R. Pan. BayesOWL: Uncertainty modeling in semantic web ontologies. *Soft Computing in Ontologies and Semantic Web*, pages 3–29. Springer–Verlag, 2006.

[3] G. Stoilos, G. Stamou, V. Tzouvaras, J.Z. Pan, and I. Horrocks. Fuzzy OWL: Uncertainty and the semantic web. In *Proc. of the International Workshop on OWL: Experiences and Directions*, volume 280. CEUR, 2005.

[4] D. Preuveneers and Y. Berbers. Quality extensions and uncertainty handling for context ontologies. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*. IOS Press, 2006.

[5] D. Brickley, R.V. Guha, and B. McBride. RDF vocabulary description language 1.0: RDF schema. *W3C recommendation*, 10:27–08, 2004.

[6] D.L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10, 2004.

[7] J. Hainaut, J. Hick, V. Englebert, J. Henrard, and D. Roland. Understanding Implementations of IS-A Relations. In B. Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling, Cottbus, Germany, October 7-10, 1996, Proceedings*, volume 1157 of *Lecture Notes in Computer Science*, pages 42–57. Springer–Verlag, 1996.

[8] C. Atkinson and T. Kühne. The essence of multilevel metamodeling. In *Proceedings of the 4th International Conference on The Unified Modeling Language Modeling Languages Concepts and Tools*, volume 2185, pages 19–33. Springer–Verlag, 2001.

[9] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0. *W3C recommendation*, 6, 2000.

[10] D. Beckett and B. McBride. RDF/XML syntax specification (revised). *W3C recommendation*, 10, 2004.

[11] J. Clark, S. DeRose, et al. XML path language (XPath) version 1.0. *W3C recommendation*, 16:1999, 1999.

[12] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):345–357. Elsevier, 2004.

[13] T. Berners-Lee and D. Connolly. Notation 3 (N3) A readable RDF syntax. *W3C Submission, Jan*, 2008.

[14] J.J. Carroll and P. Stickler. TriX: RDF triples in XML. Technical Report HPL-2004-56, Hewlett–Packard, 2004.

[15] G. Antoniou and F. Van Harmelen. *A semantic web primer*. The MIT Press, Cambridge, Massachusetts, 2008.

[16] T. Berners-Lee. Semantic web road map. `http://www.w3.org/DesignIssues/Semantic.html`, 1998.

[17] S. Sizov. What Makes You Think That? The Semantic Web's Proof Layer. *IEEE Intelligent Systems*, 22:94–99, November. IEEE Press, 2007.

[18] T. Berners-Lee. Linked Data–The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22. IGI Global, 2009.

[19] N. Drummond and R. Shearer. The open world assumption. `http://www.cs.man.ac.uk/~drummond/presentations/OWA.pdf`, 2006.

[20] R. Guha, R. McCool, and R. Fikes. Contexts for the Semantic Web. *Knowledge Creation Diffusion Utilization*, 3298(1-3):32–46. Springer–Verlag, 2004.

[21] L. Ding, T. Finin, A. Joshi, R. Pan, R.S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

[22] M.G. Butuc. Semantically Enriching Content Using OpenCalais. In *EDITIA'09: Proceedings of the Romanian Workshop on Distributed Systems, Suceava, Romania*, pages 77–88, 2009.

[23] F. Hopfgartner and J. Jose. Semantic user modelling for personal news video retrieval. *Advances in Multimedia Modeling*, pages 336–346. Springer–Verlag, 2010.

[24] C. Rizo, A. Deshpande, et al. A rapid, Web-based method for obtaining patient views on effects and side-effects of antidepressants. *Journal of Affective Disorders*, 130(1-2):290–293. Elsevier, 2011.

[25] L. Goddard and G. Byrne. Linked Data tools: Semantic Web for the masses. *First Monday*, 15(11-1). First Monday Editorial Group, 2010.

[26] T. Schandl and A. Blumauer. PoolParty: SKOS thesaurus management utilizing linked data. *The Semantic Web: Research and Applications*, pages 421–425. Springer–Verlag, 2010.

[27] T. Tudorache, J. Vendetti, and N.F. Noy. Web-Protege: A lightweight OWL ontology Editor for the Web. In *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions*. CEUR, 2008.

[28] M.C.A. Klein. *Change management for distributed ontologies*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[29] D. Lee and W.W. Chu. Comparative analysis of six XML schema languages. *ACM Sigmod Record*, 29(3):76–87. ACM, 2000.

[30] A. Souzis. Bringing the "wiki-way" to the semantic web with rhizome. In *Proceedings of the First Workshop on Semantic Wikis From Wiki To Semantics, Workshop on SemanticWikis (ESWC2006)*. CEUR, 2006.

[31] M. Sintek, L. Van Elst, S. Scerri, and S. Handschuh. Distributed knowledge representation on the social semantic desktop: named graphs, views and roles in nrl. *The Semantic Web: Research and Applications*, pages 594–608. Springer–Verlag, 2007.

[32] M. Sintek, L. van Elst, S. Scerri, and S. Handschuh. Nepomuk Representational Language Specification. `http://www.semanticdesktop.org/ontologies/nrl/`, 2007.

[33] N. Guarino and C. Welty. Evaluating ontological decisions with OntoClean. *Communications of the ACM*, 45(2):61–65. ACM, 2002.

[34] S. Auer, S. Dietzold, and T. Riechert. OntoWiki–A tool for social, semantic collaboration. *The Semantic Web-ISWC 2006*, pages 736–749. Springer–Verlag, 2006.

[35] Ó. Corcho, A. Gómez-Pérez, R. González-Cabero, and M. Suárez-Figueroa. ODEval: a tool for evaluating RDF (S), DAML+ OIL, and OWL concept taxonomies. *Artificial Intelligence Applications and Innovations*, pages 369–382. Springer–Verlag, 2004.

[36] L. Baresi, E. Di Nitto, and C. Ghezzi. Toward open-world software: Issue and challenges. *Computer*, 39(10):36–43. IEEE Press, 2006.

[37] E. Hyvönen, E. Mäkelä, T. Kauppinen, O. Alm, J. Kurki, T. Ruotsalo, K. Seppälä, J. Takala, K. Puputti, H. Kuittinen, et al. CultureSampo: A National Publication System of Cultural Heritage on the Semantic Web 2.0. *The Semantic Web: Research and Applications*, pages 851–856. Springer–Verlag, 2009.

[38] J. Laitio and O. Alm. Vera - Validation and quality assistant for Semantic Web data. `http://www.seco.tkk.fi/services/vera/`, 2008.

[39] G. Lausen, M. Meier, and M. Schmidt. SPARQLing constraints for RDF. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 499–509. ACM, 2008.

[40] P.A. Bernstein, A.Y. Halevy, and R.A. Pottinger. A vision for management of complex models. *ACM Sigmod Record*, 29(4):55–63. ACM, 2000.

[41] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. *Mechanizing Mathematical Reasoning*, pages 228–248. Springer–Verlag, 2005.

[42] A. Powell, M. Nilsson, A. Naeve, P. Johnston, and T. Baker. DCMI Abstract Model. `http://dublincore.org/documents/abstract-model/`, 2007.

[43] D. Bearman, G. Rust, S. Weibel, E. Miller, and J. Trant. A Common Model To Support Interoperable Metadata: Progress Report on Reconciling Metadata Requirements from the Dublin Core and INDECS/DOI Communities. *D-Lib Magazine*, 5:n1. CNRI, 1999.

[44] S. Decker, S. Melnik, F. Van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: The roles of XML and RDF. *Internet Computing*, 4(5):63–73. IEEE Press, 2000.

[45] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, and M. Aronson. Extending UML to support ontology engineering for the semantic web. *UML 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pages 342–360. Springer–Verlag, 2001.

[46] E. Watkins and D. Nicole. Named graphs as a mechanism for reasoning about provenance. *Frontiers of WWW Research and Development-APWeb 2006*, pages 943–948. Springer–Verlag, 2006.

[47] O. Hartig. Provenance information in the web of data. In *Proceedings of the 2nd Workshop on Linked Data on the Web (LDOW2009)*. CEUR, 2009.

[48] R. Volz, S. Handschuh, S. Staab, L. Stojanovic, and N. Stojanovic. Unveiling the hidden bride: deep annotation for mapping and migrating legacy data to the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(2):187–206. Elsevier, 2004.

[49] R. Sinkkilä, O. Suominen, and E. Hyvönen. Automatic Semantic Subject Indexing of Web Documents in Highly Inflected Languages. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011)*. Springer–Verlag, June 2011.

[50] M. Frosterus, E. Hyvönen, and Mika Wahlroos. Extending Ontologies with Free Keywords in a Collaborative Annotation Environment. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*. Springer–Verlag, October 2011. Accepted for publication.

[51] C. Heath and A. Tversky. Preference and belief: Ambiguity and competence in choice under uncertainty. *Journal of Risk and Uncertainty*, 4(1):5–28. Springer–Verlag, 1991.

[52] A. Tversky and D. Kahneman. Judgment under uncertainty: Heuristics and biases. *Judgment and decision making: An interdisciplinary reader*, page 35. Cambridge University Press, 2000.

[53] J. Kurki and E. Hyvönen. Collaborative Metadata Editor Integrated with Ontology Services and Faceted Portals. In *Workshop on Ontology Repositories and Editors for the Semantic Web (ORES 2010), the Extended Semantic Web Conference ESWC 2010*. CEUR, June 2010.

[54] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J.A. Tomlin, et al. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the 12th international conference on World Wide Web*, pages 178–186. ACM, 2003.

[55] O. Medelyan. *Human-competitive automatic topic indexing*. PhD thesis, The University of Waikato, 2009.

[56] O. Alm. Tekstidokumenttien automaattinen ontologiaperustainen annotointi. Master's thesis, University of Helsinki, Department of Computer Science, September 2007.

[57] I.H. Witten, G.W. Paynter, E. Frank, C. Gutwin, and C.G. Nevill-Manning. KEA: Practical automatic keyphrase extraction. In *Proceedings of the fourth ACM conference on Digital libraries*, pages 254–255. ACM, 1999.

[58] O. Medelyan and I.H. Witten. Thesaurus based automatic keyphrase indexing. In *Digital Libraries, 2006. JCDL'06. Proceedings of the 6th ACM/IEEE-CS Joint Conference on*, pages 296–297. IEEE Press, 2006.

[59] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005.

[60] K. Viljanen, J. Tuominen, and E. Hyvönen. Ontology Libraries for Production Use: The Finnish Ontology Library Service ONKI. In *Proceedings of the 6th European Semantic Web Conference (ESWC 2009)*. Springer–Verlag, May 31 - June 4 2009.

[61] E. Prud'Hommeaux and A. et al. Seaborne. SPARQL query language for RDF. *W3C Recommendation*, 4, 2008.

[62] E. Mäkelä, K. Hypén, and E. Hyvönen. BookSampo—Lessons Learned in Creating a Semantic Portal for Fiction Literature. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*. Springer–Verlag, 2011.

[63] T. Groza, S. Handschuh, K. Moeller, G. Grimnes, L. Sauermann, E. Minack, C. Mesnage, M. Jazayeri, G. Reif, and R. Gudjonsdottir. The NEPOMUK project - on the way to the social semantic desktop. *Proceedings of I-Semantics*, 7:201–211. JUCS, 2007.

[64] P.V. Biron, A. Malhotra, et al. XML schema part 2: Datatypes. *W3C recommendation*, 2:2–20010502, 2001.

[65] E. Mäkelä and E. Hyvönen. How to deal with massively heterogeneous cultural heritage data – lessons learned in CultureSampo. *Semantic Web – Interoperability, Usability, Applicability*. IOS Press, 2011.

[66] O. Valkeapää and E. Hyvönen. A Browser-based Tool for Collaborative Distributed Annotation for the Semantic Web. In *5th International Semantic Web Conference, Semantic Authoring and Annotation Workshop*, November 2006.

[67] O. Valkeapää, O. Alm, and E. Hyvönen. Efficient content creation on the semantic web using metadata schemas with domain ontology services (system description). *The Semantic Web: Research and Applications*, pages 819–828. Springer–Verlag, 2007.

Appendix

Generic XML to RDF transformer in Java

Below is the source code for a generic XML to RDF transformer, written in the Java programming language. It takes any XML file as input and provides an RDF model as output. It does not make any assumptions of the source format, so it is guaranteed to work on any kind of XML, though the quality of the output depends on how well the source data translates into triple-based data.

```
import java.io.File;
import java.io.FileReader;
import java.util.Stack;

import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.vocabulary.RDF;
import com.hp.hpl.jena.vocabulary.RDFS;

/**
 *
 * A generic XML to RDF transformer.
 * Depends on Apache Xerces for XML and
 * the Jena framework for RDF manipulation
 *
 * @author Joonas Laitio
 */
public class XMLToRDFParser extends DefaultHandler
{
    /**
     * Controls if empty elements or elements containing only whitespace should
     * be generated into empty literal triples in the RDF output.
     */
    private static final boolean makeEmptyLiteralTriples = false;


    private final String NS;
    private Model m;
    private Stack<Resource> elementStack = new Stack<Resource>();

    private int serial = 0;

    private String lastOpened;
    private StringBuilder lastValue = null;

    private XMLToRDFParser(String prefix, String uri)
    {
        this.m = ModelFactory.createDefaultModel();
        this.m.setNsPrefix(prefix, uri);
        this.NS = uri;
    }

    @Override
    public void startElement(String uri, String localName,
        String qName, Attributes attributes)
```

```
throws SAXException
{
    this.lastOpened = qName;

    if (serial % 10000 == 0)
        System.out.print(".");
    if (serial % 1000000 == 0)
        System.out.println();

    // Create a new resource for this element (if this element happens
    // to contain only a text node, no triples are ever made for this
    // resource)
    Resource r = m.getResource(NS + qName
        + Long.toHexString(System.currentTimeMillis())
        + Long.toHexString(++serial));

    if (attributes.getLength() > 0)
    {
        for (int i = 0 ; i < attributes.getLength() ; i++)
            m.add(r, m.getProperty(NS + attributes.getLocalName(i)),
                attributes.getValue(i));
    }

    this.elementStack.push(r);

    this.lastValue = new StringBuilder();
}

@Override
public void characters(char[] ch, int start, int length)
throws SAXException
{
    String value = new String(ch, start, length);

    if (value.trim().isEmpty())
        return;

    this.lastValue.append(value);
}

@Override
public void endElement(String uri, String localName,
    String qName)
throws SAXException
{
    if (lastOpened.equals(qName) && lastValue != null &&
        this.elementStack.size() > 1)
    {
        // Literal value
        this.elementStack.pop();
        String value = this.lastValue.toString();
        if (!value.trim().isEmpty() || makeEmptyLiteralTriples)
            m.add(this.elementStack.peek(), m.getProperty(NS + qName), value);
        this.lastValue = null;
    }
    else if (!lastOpened.equals(qName) && this.elementStack.size() > 1)
    {
        // Object value
        Resource was = this.elementStack.pop();
        m.add(this.elementStack.peek(), m.getProperty(NS + "has_" + qName),
            was);
        m.add(was, RDF.type, m.getResource(NS + qName));
    }
    else if (this.elementStack.size() <= 1)
        this.elementStack.pop();

    this.lastValue = new StringBuilder();
}
```

```
    private Model getModel()
    {
        return this.m;
    }

    /**
     * Transforms an XML file into RDF
     *
     * @param file The XML file
     * @param prefix Desired namespace prefix for the document (e.g. "seco-core")
     * @param uri The URI corresponding to the above namespace prefix
     *        (e.g. "http://www.yso.fi/onto/seco-core/")
     * @return Jena Model of the generated RDF
     */
    public static Model parse(File file, String prefix, String uri)
    {
        XMLToRDFParser parser = new XMLToRDFParser(prefix, uri);

        try
        {
            InputSource data = new InputSource(new FileReader(file));

            SAXParser saxParser = new SAXParser();
            saxParser.setContentHandler(parser);
            saxParser.parse(data);

        }
        catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println();

        parser.getModel().setNsPrefix("rdf", RDF.getURI());
        parser.getModel().setNsPrefix("rdfs", RDFS.getURI());

        return parser.getModel();
    }
}
```